Fundamental Limits of Shared-Cache Networks

Emanuele Parrinello

SORBONNE UNIVERSITÉ

EURECOM
Sophia Antipolis

Fundamental Limits of Shared-Cache Networks

Dissertation

submitted to

Sorbonne Université

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Author:

Emanuele PARRINELLO

Thesis Supervisor: Prof. **Petros ELIA**

Scheduled for defense on the 23 SEPTEMBER, 2021, before a committee composed of:

Reviewers

Prof.	**Daniela TUNINETTI**	University of Illinois at Chicago
Prof.	**B. Sundar RAJAN**	Indian Institute of Science

Examiners

Prof.	**Giuseppe CAIRE**	Technical University of Berlin
Prof.	**Marios KOUNTOURIS**	EURECOM
Prof.	**Mari KOBAYASHI**	CentraleSupelec
Prof.	**Daniela TUNINETTI**	University of Illinois at Chicago
Prof.	**B. Sundar RAJAN**	Indian Institute of Science

SORBONNE UNIVERSITÉ

EURECOM
Sophia Antipolis

Limites Fondamentales des Réseaux avec Caches Partagés

Thèse

soumise à

Sorbonne Université

pour l'obtention du Grade de Docteur

présentée par:

Emanuele PARRINELLO

Directeur de thèse: Prof. **Petros ELIA**

Soutenance de thèse prévue le 23 SEP 2021 devant le jury composé de:

Rapporteurs

Prof.	**Daniela TUNINETTI**	University of Illinois at Chicago
Prof.	**B. Sundar RAJAN**	Indian Institute of Science

Examinateurs

Prof.	**Giuseppe CAIRE**	Technical University of Berlin
Prof.	**Marios KOUNTOURIS**	EURECOM
Prof.	**Mari KOBAYASHI**	CentraleSupelec
Prof.	**Daniela TUNINETTI**	University of Illinois at Chicago
Prof.	**B. Sundar RAJAN**	Indian Institute of Science

Abstract

In the context of communication networks, the emergence of predictable content has brought to the fore the use of caching as a fundamental ingredient for handling the exponential growth in data volumes. At the same time, an information theoretic study of cache-aided networks, commonly known as *coded caching*, has revealed the powerful benefits of combining caching and coded multicasting in allowing for the elusive scaling of networks. This new caching approach has generated great interest in the research community, which has done significant work both in terms of novel achievable schemes and converse bounds for several different cache-aided scenarios. Nevertheless, a lot of effort is still required towards studying cache-aided settings that captures the essence of more realistic wireless communication networks, with all their heterogeneities, topological characteristics, fundamental bottlenecks and practical limitations that often arise from the random nature of such networks.

One of the major focuses of this thesis is the study of the fundamentals of shared-cache networks where the communication to users is aided by a small set of caches, each serving a potentially arbitrary number of users. Our addressed shared-cache setting, not only captures heterogeneous wireless cellular networks where cache-aided small base stations coexist with macro base stations, but it can also represent a model for users requesting multiple files simultaneously. Furthermore, limiting the number of caches to a much smaller value than the number of users might be inevitable in the presence of the crippling subpacketization bottleneck of coded caching. That is why we believe that the study of the shared-cache setting is of critical importance in the development of the theory and practice of coded caching. For such networks, we will characterize under some reasonable assumptions the optimal normalized delivery time required to serve all the users, both in the presence and absence of knowledge of the network topology (i.e. number of users associated to each cache) during the cache placement phase. We will also reveal that properly allocating cache memory as a function of the topology is a crucial ingredient for achieving the fundamental limits of such networks, even in those cases when it is not possible, during the caching phase, to acquire the knowledge of the exact topology. In order to show the versatility of our results, we will employ the techniques developed for the shared-cache networks in the context of coded distributed computing with heterogeneous resources, for which some information theoretic guarantees will be also provided. Furthermore, in line with the current trends of employing large antenna arrays at the base stations, this thesis also studies the caching setting where the main transmitter

is equipped with multiple antennas, for which we will show exactly how the optimal linear sum degrees of freedom (DoF) evolves as a function of the number of antennas, the number of caches and the topology of the network. At the same time, we will also propose a novel multi-antenna shared-cache-based scheme which has low complexity both in terms of subpacketization requirements and optimized beamforming design, thus allowing to serve a large number of cache-aided users with improved rate performance compared to the classical uncoded approach. Finally, the thesis will also touch upon the topic of coded caching with users involving heterogeneous Quality-of-Service (QoS) requirements, where the derived results nicely show how the development of tight converse bounds can allow the insightful identification of the optimal caching strategy. In the end, this thesis provides novel information theoretic converses and schemes for a class of settings that we believe are crucial in the evolution of cache-aided communications. We hope that the new tools that we developed while constructing these bounds and coding schemes, can be of use in various future efforts that necessitate the transition from the simplifying homogeneity and symmetry to the heterogeneity and asymmetry that are indeed at the core of modern communication networks.

Acknowledgements

I want to truly thank a large set of people who helped and supported me during the journey that lead to this Ph.D. thesis.
First of all, I want to express my deep gratitude to my supervisor Prof. Petros Elia for giving me the opportunity to pursue this Ph.D. and for his continuous support during this years. His constructive criticism, along with his deep understanding of the topic has definitely helped me to successively improve my research work as well as my way of working/thinking. This Ph.D. work would have not proceeded so fast if he had not been so generous with his time.
I want to thank all my co-authors, Ayse, Berksan, Adeel, Eleftherios, Antonio, Mohammad, Pooya and Antti, with whom I collaborated during these years. Their different way of working, knowledge and opinions have greatly contributed to this thesis and helped me to grow scientifically.
I am also grateful to all the committee members, Prof. Tuninetti, Prof. Rajan, Prof.Caire, Prof. Kountouris and Prof. Kobayashi for their time and insightful comments. It was an honor to present my work to them.
This Ph.D. work concludes my time at EURECOM, which proved to me to be an amazing work environment where I have met and I have got to know many nice people, with whom I have been spending a lot of time even outside EURECOM's walls. I think very few people have in life the privilege to share the working environment with their closest friends. That is why I am extremely grateful to my friends Lorenzo, Placido but also Jonas and Sagar for the time we spent together and their encouraging words during some difficult times of my Ph.D. life. Big thanks go also to my friend Tryfon for the joyful time that we have spent together.
To my mum, dad and brother goes my deepest gratitude for their endless love and constant support. Last but not least, I want to thank immensely Marta, for her love, support and patience. Her presence at home and the joyful time spent together have significantly helped me in the last part of my Ph.D.

Contents

Acronyms and Abbreviations

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

AWGN	Additive White Gaussian Noise.
BC	Broadcast Channel.
CDN	Content Delivery Network.
CSIT	Channel State Information at the Transmitter.
D2D	Device to Device.
DoF	Degrees of Freedom.
IoT	Internet of Things.
IP	Internet Protocol.
ISP	Internet Service Providers,
LHS	left hand side.
MAC	Multiple Access Channel.
MAN	Maddah-Ali and Niesen.
MAIS	Maximum Acyclic Induced Subgraph.
MBS	Macro Base Station.
MIMO	Multiple Input Multiple Output.
MISO	Multiple Input Single Output.
MMSE	Minimum Mean Square Error.
NDT	Normalized Delivery Time.
QoS	Quality of Service.
RIS	Reconfigurable Intelligent Surfaces.
RHS	right hand side.
RX	Receiver.
RZF	Regularized Zero-Forcing.
SBS	Small Base Station.
SCA	Successice Convex Approximation.
SIMO	Single Input Multiple Output.
SINR	Signal to Noise and Interference Ratio.
SNR	Signal-to-Noise Ratio.
SISO	Single-Input Single-Output.
TDMA	Time Division Multiple Access.
TX	Transmitter.

UL	Uplink.
w.l.o.g.	Without loss of generality.
ZF	Zero Forcing.

Notations

Vectors are denoted by bold symbols. We use the notation $\mathbf{v}(i)$ to denote the i-th entry of vector $\mathbf{v}$. Whenever needed, sets are considered to be ordered such that $\mathcal{T}(i)$ denotes the i-th element of set $\mathcal{T}$.

$\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{C}$	denote the sets of natural, integer, real and complex numbers, respectively
$[n]$	denotes the set $\{1, 2, \ldots, n\}$, i.e. $[n] \triangleq \{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$
$[n]_0$	denotes the set $\{0, 1, 2, \ldots, n\}$, i.e. $[n]_0 \triangleq \{0, 1, 2, \ldots, n\}$ for $n \in \mathbb{N}$
$2^{\mathcal{T}}$	denotes the powerset of the set $\mathcal{T}$
$n \vert m$	denotes that n divides integer m for $n, m \in \mathbb{N}$
$\vert\mathcal{A}\vert$	denotes the cardinality of set $\mathcal{A}$
$\mathcal{A}^c$	denotes the complement of the set $\mathcal{A}$
$\mathcal{A} \setminus \mathcal{B}$	denotes the set difference of sets $\mathcal{A}, \mathcal{B}$, i.e. $\mathcal{A} \setminus \mathcal{B} \triangleq \{\tau \in \mathcal{A} \wedge \tau \neq \mathcal{B}\}$
$C_k^{\mathcal{T}}$	denotes the set of all k-combinations of set $\mathcal{T}$, i.e. $C_k^{\mathcal{T}} \triangleq \{\tau : \tau \subseteq \mathcal{T}, \vert\tau\vert = k\}$
$P(n, k)$	denotes the number of k-permutations of n elements, i.e. $P(n, k) \triangleq \frac{n!}{(n-k)!}$ for $n, k \in \mathbb{N}$
$\binom{n}{k}$	denotes the number of k-combinations of n elements, i.e. $\binom{n}{k} \triangleq \frac{n!}{(n-k)!k!}$ for $n, k \in \mathbb{N}$
$\mathbf{v} \Vert \mathbf{w}$	denotes the concatenation of vector $\mathbf{v}$ with vector $\mathbf{w}$
$(\mathbf{v} \Vert \mathbf{v})_N$	denotes the concatenation of a vector $\mathbf{v}$ with itself N times
$\bigoplus$	denotes the XOR operation
$(\cdot)^T$	denotes the transpose operator
$(\cdot)^H$	denotes the conjugate transpose operator
round$(\cdot)$	denotes the function rounding a real number to the nearest integer
$gcd(\cdot)$	is the function that gives the greatest common divisor of the input arguments.
$\text{Conv}_{k \in \mathcal{X}}(\text{a}_k)$	is a continuous real function that denotes the lower convex envelope of the points $\{(k, \text{a}_k) \vert k \in \mathcal{X}\}$

In each chapter we will introduce more notation that remains confined to that chapter.

Chapter 1

Introduction to Coded Caching

1.1 Motivation for Caching

In recent years, the rapid increase and evolution of Internet devices and high bandwidth-consuming applications has brought an unprecedented exponential growth of mobile data traffic. In the near future, the explosion of Internet of Things (IoT) and new data-hungry applications (e.g. Virtual Reality) relying on machine learning and big data will make the growth of mobile data traffic even more alarming. According to a Cisco [1], in 2022 the mobile data traffic will be seven times higher than the amount in 2017. At the same time, again according to Cisco forecast (Figure 1.1), more than 80% of all IP traffic will be video in 2022. We can safely conclude that, video traffic is driving global traffic growth. Many new promising technologies that aim at increasing the capacity of current mobile networks, such us multi-cell coordination, network densification, Multiple-Input Multiple-Output (MIMO) systems, millimeter-waves communication and Reconfigurable Intelligent Surfaces (RIS) may indeed offer some promising directions, but non exploit the properties of video traffic.

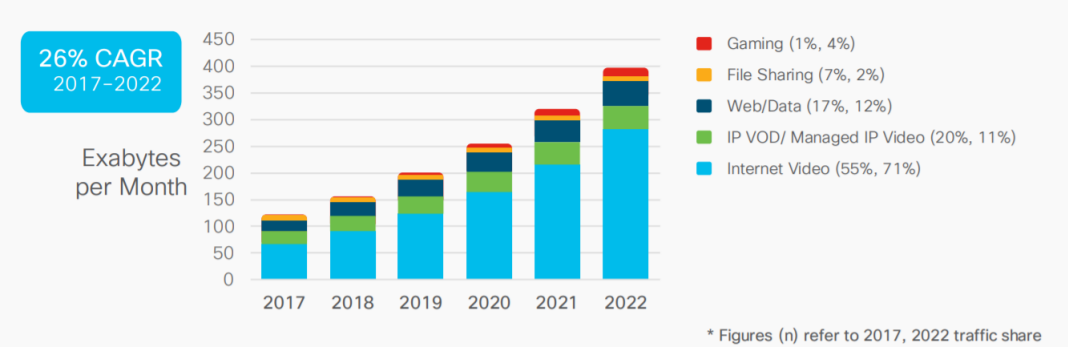


Figure 1.1 – Source: Cisco VNI Global IP Traffic Forecast, 2017–2022

On-Demand video traffic has a few peculiar characteristics which, if properly exploited, can significantly help Internet Service Providers (ISP) to reduce their network traffic. First of all, video content is often not generated in real time and it can therefore be stored

and replicated several times in the networks. Secondly, video traffic is characterized by a high temporal variability which makes the network congested during the peak hours and overprovisioned during off-peak times. These attributes and the fact that users' demands are often limited to a relatively restricted number of popular files are crucial ingredients that allow *caching* to be one of the most promising solutions toward reducing congestion in current and future communication networks. These benefits of caching are compounded by the still ever increasing storage capabilities of modestly sized devices. Caching allows us to translate Moore's law in storage capacities into gains for wired and wireless networks.

1.1.1 Classical Caching Systems

In content delivery networks (CDNs), caching is a technique that consists in bringing, during the off-peak hours, the most popular content closer to the users. This has been traditionally done by replicating this content in storage devices distributed across the network. In doing so, caching reduces the load of ISP's networks during the peak-hours, it balances the traffic of the network over time and it reduces the delay for the end users. For example, Netflix has been using caching since 2011, when it started to build its own CDN. Nowadays, almost all Netflix content is served from Netflix-operated caches installed in the networks of local ISPs. Other examples of CDNs are those by Google, Akamai and Facebook, while additional examples of different caching systems are the Domain Name System (DNS), browser caching, computer memory caches and so on. This traditional caching approach essentially reduces the amount of data requested to the original server storing the content, thus resulting in what we will refer to as *local caching gain*.

In the last years, information theoretic studies on cache networks have revealed some fundamental insights that have debunked the conventional beliefs about caching that were keeping locked the true potentials of these networks. In fact, while it was common to believe that the usefulness of caching is limited to bringing content closer to the users, different ground-braking information theoretic studies have revealed that the main gain from using cache networks stems from their ability to communicate content globally. These fundamental insights were brought to light in the seminal work by Maddah-Ali and Niesen (MAN) in [2], where the authors proposed *Coded Caching* as a powerful technique that is able to unleash so-called *global caching gains*.

1.2 Coded Caching

1.2.1 Cache-Aided Shared-Link Broadcast Channel

Coded Caching was proposed for the first time in [2], for a network where a server with access to a library of N files serves — through a noiseless shared-link channel of capacity 1 file per unit of time — a set of K users, each equipped a storage unit of size M (in units of files), so that each user can store a fraction $\gamma \triangleq \frac{M}{N}$ of the library. The system

is assumed to operate in two different and sequential phases. The first, referred to as *cache placement phase*, consists in filling — during the off-peak hours – the caches with a fraction of the content of the library without knowledge of the future users' demands. The second phase, called *delivery phase*, occurs when each user in the system requests a (potentially different) file from the library. The server, then, depending on the requested files and the content cached at the users, transmits a codeword of duration T, which will be used by each user, along with the cached content, to recover its requested file. For this setting, Maddah-Ali and Niesen proposed a cache placement phase that consists of splitting each file of the library into very small chunks and carefully placing them in the users' caches according to a specific combinatorial pattern. This novel cache placement induces in the delivery phase coded multicasting opportunities that are properly exploited by a coding scheme that delivers XORs that serve $K\gamma + 1$ users simultaneously. The work in [2] has shown that irrespective of the user demands, the normalized delivery time (NDT) cannot be worse than

$$T^*_{MAN} = \frac{K(1-\gamma)}{K\gamma + 1}, \qquad \gamma \in \left\{0, \frac{1}{K}, \frac{2}{K}, \ldots, 1\right\}. \tag{1.1}$$

If we have a careful look at the NDT in (1.1), we can identify two important quantities. First of all, we identify the numerator term $(1-\gamma)$, which is commonly referred to as *local caching gain* and which tells us how much of each file has to be transmitted to each user in additional to the amount that the user has in its own cache. Then, we identify the denominator term $K\gamma + 1$, which is usually referred to as *global caching gain* and which represents the number of users that can be served simultaneously in the delivery phase. We will also refer to this quantity as the *sum degrees of freedom* of the network, which is here defined as

$$DoF \triangleq \frac{K(1-\gamma)}{T}, \tag{1.2}$$

reflecting the rate of delivery of the non-cached desired information. This achievable performance was proved to be information theoretic optimal within a multiplicative factor of 2 in [3] and exactly optimal under the assumption of *uncoded* cache placement in [4] (see also [5]). Here, the term *uncoded cache placement* refers to any cache placement strategy that stores the bits of the library in the caches without applying any coding. It is important to point out that $T^*_{MAN}(K,t)$ is optimal in terms of the worst-case performance, while a smaller delivery time can be achieved when a subset of users have identical requests [5]. Furthermore, the MAN scheme [2] requires that during the placement phase the server be already aware of the number and identity of the users that will use the system in the subsequent delivery phase. For this reason, this scheme is classified as a *centralized* scheme. In some practical scenarios, the identity of the users might not be available to the server, which is forced to employ a so-called *decentralized* scheme [6]. Before presenting the general scheme achieving the above performance in (1.1), we present a simple example that helps to convey the idea of the general algorithm in [2].

A Toy Example

Consider a caching network where a server with a library of $N = 4$ equally-sized files $W^{(1)}, W^{(2)}, W^{(3)}, W^{(4)}$ serves $K = 4$ users, each equipped with a cache of size $M = 2$. We assume that the channel between the server and the users is noiseless with capacity equal to one file per unit of time. In the cache placement phase, the MAN scheme in [2] splits each file of the library in 6 equally-sized subfiles as follows:

$$W^{(n)} = \{W_{12}^{(n)}, W_{13}^{(n)}, W_{14}^{(n)}, W_{23}^{(n)}, W_{24}^{(n)}, W_{34}^{(n)}\},$$

where each subfile $W_{\tau}^{(n)}, n \in [N], \tau \subset [K], |\tau| = 2$ has size/duration $|W_{\tau}^{(n)}| = \frac{1}{6}$. Then, each user $k \in [4]$ fills its cache $\mathcal{Z}_k$ in the following way:

$$\mathcal{Z}_1 = \{W_{12}^{(n)}, W_{13}^{(n)}, W_{14}^{(n)} \ \forall n \in [4]\},$$

$$\mathcal{Z}_2 = \{W_{12}^{(n)}, W_{23}^{(n)}, W_{24}^{(n)} \ \forall n \in [4]\},$$

$$\mathcal{Z}_3 = \{W_{13}^{(n)}, W_{23}^{(n)}, W_{34}^{(n)} \ \forall n \in [4]\},$$

$$\mathcal{Z}_4 = \{W_{14}^{(n)}, W_{24}^{(n)}, W_{34}^{(n)} \ \forall n \in [4]\}.$$

We observe that, because of the above cache placement, when each user will place a request to the server, she will have to retrieve from the library only 3 out of the 6 subfiles that compose the requested file, regardless of the specific request. Let us now assume that in the delivery phase each user requests a different file so that users $1, 2, 3, 4$ request files $W^{(1)}, W^{(2)}, W^{(3)}, W^{(4)}$, respectively. Then, the MAN strategy will create and sequentially transmit the following bit-wise XORs:

$$\begin{aligned} \mathbf{x}_{123} &= W_{23}^{(1)} \oplus W_{13}^{(2)} \oplus W_{12}^{(3)}, \\ \mathbf{x}_{124} &= W_{24}^{(1)} \oplus W_{14}^{(2)} \oplus W_{12}^{(4)}, \\ \mathbf{x}_{134} &= W_{34}^{(1)} \oplus W_{14}^{(3)} \oplus W_{13}^{(4)}, \\ \mathbf{x}_{234} &= W_{34}^{(2)} \oplus W_{24}^{(3)} \oplus W_{23}^{(4)}. \end{aligned} \tag{1.3}$$

Let us now focus on the transmitted message $\mathbf{x}_{123}$. We observe that user 1 has in its own cache the subfiles $W_{13}^{(2)}$ and $W_{12}^{(3)}$, which can therefore be removed from $\mathbf{x}_{123}$ to get the desired subfile $W_{23}^{(1)}$ free of interference from the subfiles intended for users 2 and 3. Due to its cached content, user 1 can recover the other missing subfiles $W_{24}^{(1)}$ and $W_{34}^{(1)}$ from the transmitted messages $\mathbf{x}_{124}$ and $\mathbf{x}_{134}$, respectively. Following similar arguments, we can conclude that all the other users can also successfully recover all their desired subfiles from the transmitted messages in (1.3). The total normalized delivery time T needed to successfully deliver all the subfiles to the 4 users is given by the sum of the duration of the four above messages, i.e. $T^* = 4 \times \frac{1}{6} = \frac{2}{3}$. Notice that if we transmitted the 12 subfiles one by one the total delivery time would be as high as $T = 2$. Thus, coded caching allows for a multiplicative reduction in the delay by a factor of $DoF = 3$.

The MAN Coded Caching Scheme

Having given the above example, we proceed to provide the MAN placement and delivery schemes in their general form.

MAN Cache Placement Each file $W^{(n)}$ of the library is split into $S = \binom{K}{K\gamma}$ disjoint equally-sized subfiles as follows:

$$W^{(n)} = (W_\tau^{(n)} : \tau \subseteq [K] : |\tau| = K\gamma). \tag{1.4}$$

For each $n \in [N]$, subfile $W_\tau^{(n)}$ is stored in the cache of user k if and only if $k \in \tau$. It follows that the cache of user $k \in [K]$ consists of the following content

$$\mathcal{Z}_k = \{W_\tau^{(n)} : \tau \ni k, \forall n \in [N]\}. \tag{1.5}$$

Hence, each user stores in its cache a total of $N\binom{K-1}{K\gamma-1}$ subfiles each of size $\frac{1}{\binom{K}{K\gamma}}$, which account for a total used memory of size

$$N\binom{K-1}{K\gamma-1}\frac{1}{\binom{K}{K\gamma}} = M, \tag{1.6}$$

thus satisfying the per-user cache size constraint. We now proceed with the definition of the so-called subpacketization requirement of a coded caching scheme.

Definition 1. We use the term *subpacketization requirement* to refer to the number of subfiles in which a coded caching scheme has to split each file of the library.

The MAN coded caching scheme has a subpacketization requirement of $S_{MAN} = \binom{K}{K\gamma}$, which is exponential in K.

MAN Delivery Scheme Consider all the $\binom{K}{K\gamma+1}$ sets $Q \subset [K]$ of $K\gamma+1$ users. For each such set Q, the server creates a message denoted by $\mathbf{x}_Q$ and transmits it to the $K\gamma+1$ users in the set. The message $\mathbf{x}_Q$ takes the form

$$\mathbf{x}_Q = \bigoplus_{k\in Q} W_{Q\setminus\{k\}}^{(d_k)}. \tag{1.7}$$

By construction, it can be easily verified that each subfile in the XOR is desired by one of the $K\gamma+1$ users in Q and it is available in the local caches of the other $K\gamma$ users in Q.

Achievable performance Observing that the total number of transmissions is $\binom{K}{K\gamma+1}$ and that each such transmission takes $\frac{1}{\binom{K}{K\gamma}}$ units of time, immediately tells us that the total delivery time can be expressed as in equation (1.1).

1.2.2 Extensions of the MAN Coded Caching Scheme to Other Settings

The MAN scheme has sparked significant interest and the coded caching idea has been extended to a wide range of settings. For example the work in [7] extended coded caching to the so-called multi-server setting where the library is shared among N_0 servers, connected to the users through a full-rank linear network. In terms of DoF, this setting is isomorphic to the cache-aided Multiple-input Single-Output (MISO) channel studied in [8]. A hierarchical setting with two layers of caches has been studied in [9]. Coded caching under an arbitrary popularity distribution of the file library was investigated in [10, 11]. Settings with secrecy and privacy constraints were addressed in [12–14]. In [15], the authors extended the coded caching scheme to a setting where the users use a Device-to-Device (D2D) communication scheme to recover their requested files without the help of the server. Despite the relative young age of coded caching, its literature is very wide and we here mention only a few among the many relevant works.

Applications of Coded Caching In its most general sense, coded caching is a communication technique that can be used whenever the requested data can be pre-stored in the nodes of the communication network. That is why, in addition to being employed for CDNs, coded caching can be applied to a variety of problems such us the inter-server communication bottleneck of distributed computing [16], data rebalancing in distributed databases [17], distributed data shuffling [18], medical data sharing [19], and so on. A practical application of coded caching can be found in the products of the company *Cadami* [20], which has developed coded-caching-based software for media content distribution.

In what follows we will briefly elaborate more on the multi-server/multi-antenna and the D2D settings, which are the most relevant for this thesis.

Multi-Antenna/Multi-Server Coded Caching

The multi-server setting differs from the MAN shared-link setting in the way the channel connects the servers to the users. Here, the library of files is fully present at each of the N_0 servers, which communicate with the users through a linear network that can be described by a full-rank $K \times N_0$ matrix[1] $\mathbf{H}$. The work in [7] showed that the normalized delivery time to serve all the users is no larger than

$$T^*_{MS} = \frac{K(1-\gamma)}{K\gamma + N_0}, \qquad \gamma \in \left\{0, \frac{1}{K}, \frac{2}{K}, \dots, 1\right\}, \tag{1.8}$$

which corresponds to a DoF of $DoF = K\gamma + N_0$, achieved with a subpacketization requirement of $S_{MS} = \binom{K}{K\gamma} \cdot \binom{K-K\gamma-1}{N_0-1}$. The scheme achieving this performance was proved in [21] to be optimal within a gap of 2 among all one-shot linear schemes; gap that was later tightened in [22] that showed the exact optimality under the aforementioned assumptions. The above DoF reveals that the multiplexing gain N_0 from having multiple

[1] In the DoF regime, the cache-aided MISO BC in [8] is isomorphic to the multi-server setting.

servers (multiple antennas) can be additively combined with the coded caching gain $K\gamma$ [21].

D2D Coded Caching

Another relevant setting for this thesis is the cache-aided D2D setting, where the use of coded caching was introduced for the first time in [15]. There, the authors considered a cache placement phase that is not different than the one of the cache-aided shared-link setting, while the delivery phase differs in the way the users are served. In fact, the cache-aided users in the network recover their requested files by exchanging messages in a D2D fashion, without any central server/transmitter assisting them. It is evident that in order to satisfy any possible user request, the cache placement scheme has to force all the bits of the library to be cached at the users' storage units. The work in [15] proposed a D2D-version of the MAN coded caching scheme which achieves the optimal worst-case delivery time

$$T^*_{D2D} = \frac{K(1-\gamma)}{K\gamma}, \qquad \gamma \in \left\{ \frac{1}{K}, \frac{2}{K}, \dots, 1 \right\}, \tag{1.9}$$

with a subpacketization requirement of $S_{D2D} = K\gamma \binom{K}{K\gamma}$. As we will discuss at the end of this thesis, the D2D coded caching scheme in [15] finds application in distributed computing where its use allows for the reduction of the inter-server communication load in a class of computational problems that fall under the MapReduce framework [16, 23].

Chapter 2

Motivations and Main Contributions

Having presented a general introduction to the topic of coded caching, we start this chapter by motivating shared-cache networks. These networks will be the main object of study of this thesis. Then, after providing a brief description of some related state of the art, in the second part of this chapter we will provide a succinct description of the main contributions of the thesis.

2.1 The Emergence of Shared-Cache Networks

In what follows, we will discuss how the shared-cache networks studied in this thesis arise from three seemingly different problems; the problem associated to cache-aided wireless heterogeneous networks, to multiple file demands and crucially the problem of file-size constraints. We elaborate below.

Cache-Aided Heterogeneous Wireless Networks

Traditionally the use of caching in CDNs has been mostly limited to the core of the networks. However, recent developments in base station design as well as the substantial reduction in the cost of storage units, has allowed for a much more dense deployment of caches which can now take place at the level of the radio access network. These developments, together with the tremendous increase of wireless devices and data traffic, are driving an evolution away from the classical cellular networks that we now know, into the much more complex variant of wireless heterogeneous networks where different types of transmitters and cells coexist. A typical heterogeneous network scenario that is of interest here, involves the case where in the coverage area of a large macro base station (MBS), one can also find a set of small base stations (SBS) that are placed closer to the users whom — due to their proximity — they serve at higher data rates. As argued in [24] and references therein, caching at these type of networks can significantly help ISPs better operate their networks. Indeed, in our setting, these SBSs will serve the role of helper caches that store data which can be then forwarded, during peak hours, to some users at a cost that is negligible compared to the cost of sending this same information directly from the main MBS. These SBS will serve as helper caches that

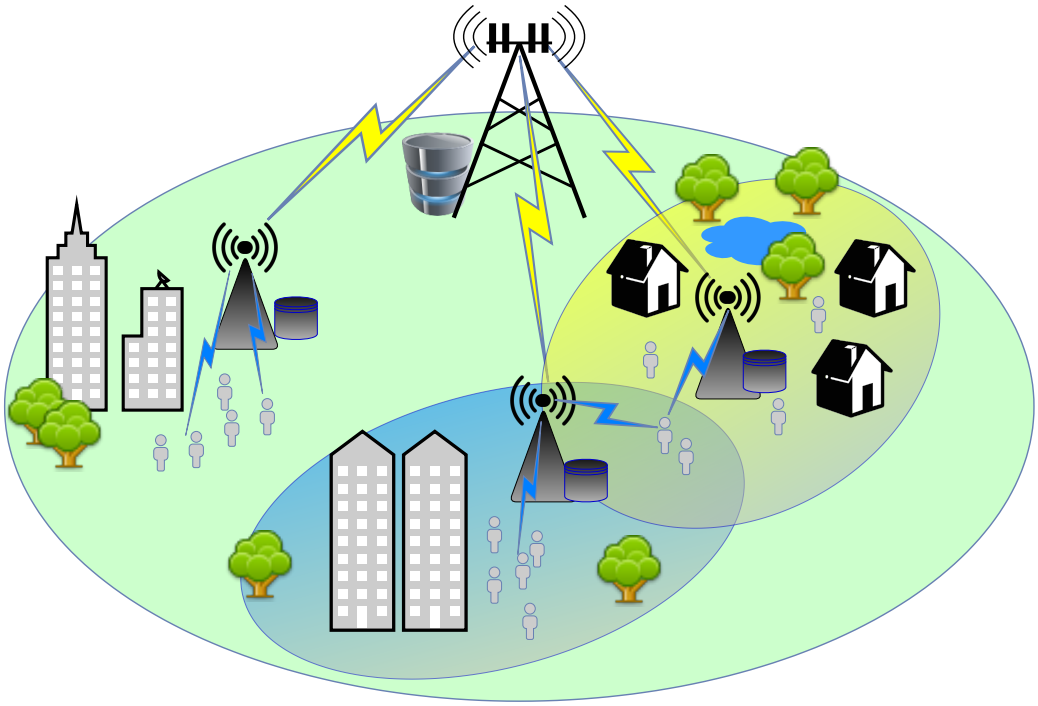


Figure 2.1 – A cache-aided heterogeneous wireless network.

are shared among various users. The transition to the coded caching setting is simple: the link between the MBS and the users corresponds to the main bottleneck link of the classical broadcast channel, and then the cache content of each user will reside in the memory of the associated SBS, i.e., will reside in the user's associated helper cache. The difference with the traditional coded caching scenario is simply that one such helper cache is associated to many users and thus these users, by necessity, share the same cache content. The connection between the heterogeneous network and the classical coded caching setting is completed after we assume that the cost of a user accessing its cache, is zero. This last assumption is motivated by the aforementioned fact that the SBS-to-user rate is expected to dwarf the corresponding rate between the MBS and the users.

Our effort will be to establish the fundamental limits of this heterogeneous cache-aided scenario. The aforementioned association between coded caching and heterogeneous networks, brings to the fore salient features that deviate from the traditional treatment of coded caching problems. One of the most challenging aspects that appears is the treatment of the heterogeneity, and in particular the uneveness in the cache occupancy number. While existing works considered a uniform setting where each cache serves the same number of users, we must in this heterogeneous setting consider networks where each cache serves an arbitrary number of users (cf. Figure 2.1). An additional related salient feature that emerges in wireless scenarios, is the need for caching schemes that are agnostic to the topology (agnostic for example of cache occupancy number). While traditional coded caching treatments assume a cache placement phase that is fully aware

of which user will be assigned which cache, it is very natural that in a realistic wireless network scenario, this knowledge may not exist. At the same time, in some other scenarios this knowledge of the network topology may be available. If not exact, in some cases it is reasonable to assume that, at the time of caching, we can have an estimate of the number of users that will be served by each cache. Therefore, studying such shared-cache networks where the cache occupancy numbers are known in the placement phase is also a very pertinent problem. Another important salient feature stems from the fact that in the wireless access setting, each user may be located in the coverage area of multiple SBSs, i.e., may have access to multiple caches. Finally but crucially, no treatment of a wireless scenario would be complete without considering that almost surely the main transmitter (the MBS) will have access to multiple transmit antennas. Treating all these salient features

- heterogeneous topology,
- users having access to multiple caches that are shared by different users,
- multiple transmit antennas,

will place interesting challenges in constructing converses as well as schemes; a challenge that will be addressed by this thesis.

Related Works on Shared-Cache Networks: One of the first studies of these cache-aided networks is the famous *Femtocaching* work in [25], where wireless receivers are assisted by helper nodes of a limited cache size, whose main role is to bring content closer to the users in order to alleviate the backhaul load from the main MBS. While [25] did not consider the use of coded caching, a transition to the latter can be found in [26] which considered a similar shared-cache network as the one in this thesis. Under the assumption that each cache serves an equal number of users, [26] proposed a coded caching scheme which was shown to perform to within a certain constant factor from the optimal. This uniform setting is addressed also in [6]. Another interesting work can also be found in [27] which explores the shared-link coded caching scenario with shared caches, where the uniformity condition is lifted, and where emphasis is placed on designing schemes with coded cache placement where the total cache size across the users is smaller than the library size. After our first results on shared-cache networks were published, other works studied similar shared-cache settings; we will briefly mention them throughout the thesis. Finally, please note that this does not capture the extent of the work that has been done in this area, and that more works are highlighted later in the text.

Coded Caching with Multiple File Requests

Assuming an error-free shared-link channel connecting the server to the users, the shared-cache setting studied in this thesis is isomorphic to a dedicated-cache setting where each receiving node equipped with its own dedicated cache, requests an arbitrary number of files. In practical scenarios, each such cache-aided users placing an arbitrary number of demands may represent a cache serving an arbitrary number of users at the edge of a

CDN. In such case, the network connecting the main server with the caches at the edge of the CDN would suffer from a bottleneck link and would then be modeled here as a error-free shared-link. Similarly, in a radio access network, a cache-aided SBS serving an arbitrary number of mobile users could forward their requests to the main server hosting the library and then, it would act as a cache-aided user with multiple file requests. In this context, under the assumption that all the users request the same number of files, interesting results have been obtained in the works in [28–30]. As we will discuss later on, the multiple file requests problem is relevant also in its D2D form which again relates to the heterogeneous coded distributed computing problem that we will discuss in Chapter 8.

Shared Caches as a Solution to the Subpacketization Bottleneck of Coded Caching

The study of the shared-cache setting is important not only because it captures the essence of some of the realistic scenarios mentioned above, but also because it nicely reflects the effect of subpacketization (file size) constraints [31] that typically force cache-aided receivers to have identically-filled caches. Unfortunately, the subpacketization requirement (see Definition 1) of coded caching algorithms is generally exponential in the number of users in the system, and thus requires unrealistically large file sizes to yield any meaningful coded caching gains. This is in our opinion one of the main bottlenecks of coded caching, and its severity is such that for any realistic file sizes, number of users and per-user cache size, the best known coding gain is a small modest single digit number, generally less than 6. At the same time, a high subpacketization requirement automatically implies a high implementation complexity and overhead [2]. That is why, in all practical scenarios, one might be forced to design a coded caching algorithm that requires that the subpacketization does not exceed a given threshold. This relates directly to the shared-cache scenario. In the single-stream setting, this connection was made in [31] which suggested the repetition of cache states (i.e., grouping users and asking the users in a group to store the same cache content) as a means of reducing subpacketization. This naturally though comes with a much reduced performance, compared to the case of unbounded file sizes. This scenario again brings to the fore the asymmetries and heterogeneities that we alluded to before. For example, in [32], Jin at al. proposed a framework where the server designs a set of Λ different cache contents according to MAN placement algorithm and where, in the cache placement phase, each user picks at random any one of such Λ caches. It is intuitive to see that, because of this decentralized cache placement, the delivery phase turns out to be isomorphic to the shared-cache setting with heterogeneously populated caches. However, the information theoretic limits of this approaches, where the number of allowed cache states is considerably smaller than the number of users, were not investigated before, and are object of study of this thesis where they are presented from the point of view of shared-cache networks. While for the cache-aided shared-link setting we will show the inevitable loss in performance of the grouping approach, a different conclusion can be drawn for the setting where the transmitter is equipped with multiple antennas. In fact, we will extend the result of Lampiris et al. that showed that, in the multiserver setting, by partitioning the set of

users in groups containing as many users as antennas (N_0) at the central server, the optimal one-shot linear DoF $K\gamma+N_0$ can still be achieved, this time with an exponentially reduced subpacketization [33]. In this work we will extend this result and show the full interplay between the number of antennas N_0 and the number of different caches Λ.

2.1.1 Storage Allocation in Cache-Aided Networks

As expected, and as we will see later on, a determining aspect to take into account when designing schemes for a scenario with heterogeneously populated caches is the ability to properly allocate storage capacity (amount of memory) at the caches as a function of their occupancy number. A SBS serving an office building is statistically more likely to serve many more users/demands than a SBS serving a sparser residential area. This heterogeneity motivates the allocation of more memory to more crowded SBSs in order to reduce the total amount of data requested to the MBS. The ability to allocate the memory as a function of the occupancy number of each cache is essential to achieve the fundamental limits of these cache-aided networks, which could not be achieved by a uniform memory allocation despite the knowledge of the occupancy number of all the caches during the cache placement phase. While one could argue that predicting during the placement phase the exact number of users that will be served by a SBS in the delivery phase can be infeasible, we will also show that a mere long-term statistical knowledge of the occupancy numbers of all the caches can still be extremely beneficial.

Related works on storage allocation: The problem of memory allocation has been investigated in the context of back-haul limited cache-aided small-cell networks [34, 35] where, however, coded multicasting was not considered. For a cache-enabled heterogeneous small-cell network, the work in [36] aims to minimize the average back-haul load subject to an overall cache capacity. However, none of the above works provides information-theoretic guarantees on the achievable performance. In contrast, we will address the memory allocation problem in a simple heterogeneous shared-cache network for which we will provide information-theoretic optimality results. The memory allocation problem was also addressed in the cache-aided degraded broadcast channel in [37] for which a memory allocation analogous to our had been proposed. A similar setting to the one in [37] can also be found in [38].

2.2 Heterogeneities in Cache-Aided Settings

Many of the problems addressed in this thesis include some heterogeneous elements. In shared-cache networks we will consider the case where each cache serves (is associated to) an arbitrary number of users, while when we address distributed computing, we will assume that each computing node has a different computational power which can lead to heterogeneous computations loads and\or unequal number of output functions to be computed by the nodes. Similarly we will study the coded caching problem where each user has a different QoS requirement. The study of such asymmetries is necessitated by the nature of communication networks, which is indeed stochastic. Any given snapshot

of a communication network reveals a variety of heterogeneities: non uniform topologies, heterogeneous resources at the user terminals, heterogeneous QoS requirements, and so on. That is why in order to understand the overall behaviour of such networks it is important to develop and analyse algorithms that can handle such heterogeneous settings. In the specific topic of cache-aided communications, such heterogeneities pose challenges in terms of the achievable schemes, which normally require complicated combinatorial designs that rely on the symmetry of the problems. These hurdles are further exacerbated when designing converse bounds, where complicated symmetrization techniques are often the key for retrieving insightful non-trivial bounds. That is why many heterogeneous coded caching problems, such us coded caching with arbitrary popularity distribution [11], coded caching with heterogeneous cache sizes [39] and coded caching with heterogeneous file sizes [40], are still largely unsolved. In this thesis we will characterize, under some reasonable assumptions, the memory-delay trade-off of some heterogeneous coded caching problems, hoping that some of our techniques and approaches could inspire the solution of other unsolved non-symmetric cache-aided problems.

2.3 Thesis Outline and Summary of the Main Contributions

This thesis seeks to study four different settings, for which the main results will be presented throughout the next chapters. In particular, the first setting studied in this thesis is the shared-link shared-cache network, which was motivated in the previous section, and which will be studied in Chapters 3,4,5. Next, we will consider the cache-aided MISO broadcast channel where the role of shared caches (and\or user grouping) is analysed in Chapter 3 and in Chapter 6. In Chapter 7 we will address the pertinent problem of coded caching with heterogeneous Quality-of-Service (QoS) requirements. After that, we will study the problem of heterogeneous coded distributed computing in Chapter 8, where we will apply the insights coming from the study of shared-cache networks. Finally, we will conclude this thesis in Chapter 9, where we will discuss the main conclusions that emerge from this work. The following provides a summarized presentation and discussion of the main results of the thesis.

2.3.1 Shared-Cache Networks

The first type of networks that we will study in this thesis is the one of a cache-aided network where each cache serves an arbitrary number of users. In particular, similarly to the cache-aided shared-link setting discussed in Section 1.2.1, a shared-link shared-cache network consists of a server with a library of N files that serves K users through an error-free shared link that can serve 1 file per unit of time. Each user is assisted at zero cost by one of the Λ ($\Lambda \leq K$) caches in the system such that — during the delivery phase — each cache $\lambda = 1, 2, \ldots, \Lambda$ serves an arbitrary set of users $\mathcal{U}_\lambda$. We will denote by L_λ the number of users associated to the λ-th most populated cache, and we will refer to the vector $\mathbf{L} = (L_1, L_2, \ldots, L_\Lambda)$ as the cache occupancy vector. Notice that in general we have that $L_\lambda \neq |\mathcal{U}_\lambda|$. We also assume that the cumulative size of the Λ caches is t times the size of the library. This setting can be separated into two major different

scenarios: the first is the topology-agnostic scenario where, during the cache placement phase, it is not known the number of users associated to each cache, and the second is the topology-aware scenario where, on the contrary, the caching phase is aware of the exact number of users assisted by each cache. Furthermore, within the context of shared-cache networks, we will seek to study the shared-cache setting where each user has access to more than one cache. Within the framework of coded caching, we will refer to this problem as the *multi-access coded caching* problem. For this setting, we will focus on the symmetric scenario with K users and $\Lambda = K$ caches, where each user has access to z different helper caches. We now proceed to present our main contributions for these aforementioned settings.

Topology-Agnostic Coded Caching with Shared Caches: In this scenario, we assume that, during the cache placement phase, the exact number of users $|\mathcal{U}_\lambda|$ associated to cache λ is not available while, instead, the cache occupancy vector $\mathbf{L}$ is known. Under the assumption of uncoded cache placement, the optimal worst-case (among all possible users' requests) normalized delivery time takes the form

$$T^*(t, \mathbf{L}) = \frac{\sum_{r=1}^{\Lambda-t} L_r \binom{\Lambda-r}{t}}{\binom{\Lambda}{t}}. \tag{2.1}$$

We will see how this optimal performance is achieved with a uniform memory allocation, i.e each cache is of size $\gamma = \frac{t}{\Lambda}$. This result allows us to conclude that, if the number of users associated to each cache is not known during the cache placement phase, any other non-uniform memory allocation cannot be helpful. What we will also see is that the NDT in (2.1) corresponds to a sum DoF smaller than $t+1$, which is achievable only if users are distributed uniformly among the caches, i.e. when $L_\lambda = \frac{K}{\Lambda}, \forall \lambda \in [\Lambda]$. The above result in (2.1) will be derived in Chapter 3 where we will present the achievable scheme and the corresponding matching converse. The latter is based on the index coding technique used in [4] which proved the optimality of the MAN normalized delivery time previously stated in equation (1.1). However, our heterogeneous setting required an interesting twist of the original converse technique that nicely allowed to push the lower bound up to the exact optimal performance. This result — which extends to the scenario where the server is equipped with a number of antennas N_0 no larger than the smallest number of users in a cache (i.e. $N_0 \leq L_\Lambda$) — can also be found in the following publications:

> [41] *E. Parrinello, A. Ünsal, and P. Elia, "Optimal coded caching in heterogeneous networks with uncoded prefetching," in IEEE Information Theory Workshop, (ITW), 2018.*
>
> [42] *E. Parrinello, A. Ünsal, and P. Elia, "Fundamental limits of coded caching with multiple antennas, shared caches and uncoded prefetching," in IEEE Transactions on Information Theory, vol. 66, no. 4, pp. 2252–2268, 2020.*

Topology-Aware Coded Caching with Shared Caches: In the considered topology-aware scenario, the exact number of users $|\mathcal{U}_\lambda|$ associated to each cache λ is known during the

cache placement phase. Without loss of generality, we here assume that cache λ is the most populated cache[1], i.e. $L_\lambda = |\mathcal{U}_\lambda|, \forall \lambda \in [\Lambda]$. This knowledge of the network topology opens up the opportunity for an optimized memory allocation as a function of $\mathbf{L}$. For each $\lambda \in [\Lambda]$, we will allocate to cache λ a (normalized) cache size

$$\gamma_\lambda \triangleq \frac{L_\lambda \cdot \sum_{q \in C_{t-1}^{[\Lambda]\setminus\{\lambda\}}} \prod_{j=1}^{t-1} L_{q(j)}}{\sum_{q \in C_t^{[\Lambda]}} \prod_{j=1}^{t} L_{q(j)}}, \quad t \in \{0, 1, \ldots, \Lambda\}, \tag{2.2}$$

where we have used the notation $C_k^{\mathcal{T}} \triangleq \{\tau : \tau \subseteq \mathcal{T}, |\tau| = k\}$. For a total cache-size budget t, we will show that all users can be served within a NDT that is no larger than

$$T(t, \mathbf{L}) = \frac{\sum_{\lambda=1}^{\Lambda} L_\lambda (1 - \gamma_\lambda)}{t + 1}, \tag{2.3}$$

where $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$ (defined above) adheres to the global cache size constraint $\sum_{\lambda=1}^{\Lambda} \gamma_\lambda = t$. Equation (2.3) directly tells us that since L_λ is the number of files requested by the users connected to cache λ, and since the quantity $(1 - \gamma_\lambda)$ corresponds to the amount of data that each user connected to cache λ has to receive, then the denominator $t + 1$ corresponds to the sum DoF of the network and reflects the number of users that can be served simultaneously. Interestingly, the memory allocation allows to achieve the sum DoF of $t + 1$ regardless of cache occupancy vector $\mathbf{L}$. This nicely deviates from the topology-agnostic scenario where this maximal DoF is achievable only when users are uniformly distributed among the caches. Furthermore, in this thesis we will show that this achievable NDT is information theoretic optimal under the assumption of uncoded and homogeneous cache placement. Here, the expression *homogeneous cache placement* refers to those placement schemes where all the bits of the library are cached the same number of times across the caches. The achievable scheme highlights the importance of memory allocation in heterogeneous settings, and the converse provides combinatorial tools that we believe can be used to develop bounds for other heterogeneous cache-aided settings. These results will be presented in Chapter 4 where we will also discuss the scenario where during the cache placement phase the available information on the topology (in particular on the cache occupancy vector) is erroneous or imperfect. In this context, we will demonstrate the benefits of memory allocation that exploits only the average cache occupancy, and how even such partial knowledge far outperforms the topology-agnostic treatment. Part of our work on topology-aware shared-cache networks resulted in the following publication

> [43] *E. Parrinello and P. Elia, "Coded caching with optimized shared-cache sizes," in 2019 IEEE Information Theory Workshop (ITW), 2019, pp. 1–5,*

and all these results will be soon submitted to the following publication:

[1] Because of the assumption that $L_\lambda = |\mathcal{U}_\lambda|$, in this topology-aware scenario, saying that the exact number of users $|\mathcal{U}_\lambda|$ associated to each cache λ is known during the cache placement phase is equivalent to saying that the cache occupancy vector $\mathbf{L}$ is known during such phase.

[44] *E. Parrinello, A. Bazco-Nogueras and P. Elia, "Fundamental limits of topology-aware shared-cache networks," to be submitted to IEEE Transactions on Information Theory, 2021.*

Multi-Access Coded Caching: Chapter 5 addresses the multi-access coded caching problem where each of the K users in the system are connected to z consecutive caches. We assume that there are in total $\Lambda = K$ caches, each with a normalized memory size $\gamma = \frac{t}{K}$, and that the topology of the network follows a cyclic-shift pattern (see Figure 5.1), where for example user K is connected to caches $\{K, 1, 2, \dots, z-1\}$. For this setting the work in [26] showed that the following worst-case NDT

$$T(t, z) = \frac{K(1 - z\gamma)}{t + 1} \tag{2.4}$$

is achievable, thus showing an increased local caching gain of $(1 - z\gamma)$ compared to the MAN setting where each user has access to only one cache ($z = 1$). What the above though was not able to show was an increase in the global caching gain, which remained at $t + 1 = \Lambda\gamma + 1 = K\gamma + 1$. In our work we will show that for the special case where $z = \frac{K-1}{t}$ the NDT

$$T\left(t, \frac{K-1}{t}\right) = \frac{K - zt}{zt + 1} = \frac{1}{K} \tag{2.5}$$

is achievable, and it is optimal under the assumption of uncoded cache placement. This shows an ability of the scheme to serve $zt + 1$ users at the same time as if there were zK caches in the system and each user had exclusive access to z of them. Furthermore, when $t = 2$ we show that a global caching gain greater than $t + 1 = 3$ is achievable while maintaining the full local caching gain $(1 - z\gamma)$. In other words, for any $z \in [K - 1]$ and $t = 2$ our achievable NDT satisfies

$$\frac{K - 2z}{4} < T(2, z) \leq \frac{K - 2z}{3}. \tag{2.6}$$

These results were published in the following publication:

[45] *B. Serbetci, E. Parrinello, and P. Elia, "Multi-access coded caching: gains beyond cache-redundancy," in 2019 IEEE Information Theory Workshop (ITW), 2019, pp.1–5.*

2.3.2 Cache-Aided MISO Broadcast Channel

The second setting of interest that we consider in this thesis assumes that the central transmitter (server) is equipped with N_0 antennas and the channel between the transmitter and the users is modeled as a Gaussian MISO flat-fading channel. Similarly as before, we assume that each user is assisted at zero cost by one of the Λ caches in the networks so that each cache λ serves $|\mathcal{U}_\lambda| = L_\lambda$ users and that the total normalized cache size is t.

In Chapter 3 we will prove that, in the topology-agnostic scenario, the worst-case NDT[2]

$$T^*(t, \mathbf{L}, N_0) = \frac{\sum_{r=1}^{\Lambda-t} L_r \binom{\Lambda-r}{t}}{N_0 \binom{\Lambda}{t}}, \tag{2.7}$$

is achievable and optimal under the assumption of uncoded cache placement as long as the number of antennas N_0 is no larger than the number of users in each cache, i.e. $N_0 \leq L_\Lambda$. This result shows the impact of the heterogeneity of the cache occupancy vector $\mathbf{L}$ — as already discussed for the single-antenna case — and it shows the regime where the gain from the multiple antennas N_0 is multiplicative. Furthermore, for the setting where each cache of size $\gamma = \frac{t}{\Lambda}$ serves exactly $\frac{K}{\Lambda}$ users, corresponding to the cache occupancy vector $\mathbf{L}_{unif} = (\frac{K}{\Lambda}, \ldots, \frac{K}{\Lambda})$, our work shows that the NDT

$$T^*(t, \mathbf{L}_{unif}, N_0) = \begin{cases} \dfrac{K - K\gamma}{N_0(\Lambda\gamma + 1)} & \Lambda < \frac{K}{N_0}, \quad \text{(2.8a)} \\ \dfrac{K - K\gamma}{N_0(\frac{K}{N_0}\gamma + 1)} = \dfrac{K - K\gamma}{K\gamma + N_0} & \Lambda \geq \frac{K}{N_0}, \quad \text{(2.8b)} \end{cases}$$

is achievable[3] and optimal under some basic assumptions (see Chapter 3). It is interesting to observe that in the regime where $N_0 \leq \frac{K}{\Lambda}$, every time that we add one extra antenna at the transmitter we increase the DoF by an additive factor of $\Lambda\gamma + 1$. On the other hand, in a system where the transmitter has more than $\frac{K}{\Lambda}$ antennas, adding one extra antenna provides us with a DoF increase of only one additional unit. A complete presentation and discussion of the above contributions can be found in Chapter 3. Part of these results have been published in the previously mentioned journal publication in [42] and part in the following conference publication

> [46] *E. Parrinello, P. Elia, and E. Lampiris, "Extending the optimality range of multi-antenna coded caching with shared caches," in 2020 IEEE International Symposium on Information Theory (ISIT), 2020, pp. 1675–1680.*

The above results can also be interpreted in the corresponding dedicated-cache setting where each user has its own physical cache, and where Λ represents the number of groups such that the users in the same group store the same content in their caches. In this scenario, the aforementioned result not only tell us the optimal NDT but very importantly it tells us that forcing a number of groups Λ smaller than $\frac{K}{N_0}$ does not allow to achieve the optimal linear one-shot DoF $K\gamma + N_0$ of the network. Selecting $\Lambda = \frac{K}{N_0}$ and placing N_0 users in each group we retrieve the result of Lampiris et al. in [33], which showed that the DoF of $K\gamma + N_0$ can be achieved with a subpacketization of $\binom{K/N_0}{K\gamma/N_0}$, thus exponentially reducing the subpacketization required by the multi-server scheme discussed in Section 1.2.2. However, the scheme in [33] requires that $N_0 \leq K\gamma$, which might not be the case in some practical scenarios where the number of antennas employed at the transmitter is very high. For the opposite regime where $N_0 \geq K\gamma$ we will show

[2] The expression in (2.7) is valid in the DoF regime.

[3] The achievability in (2.8b) holds for N_0 being an integer multiple of $\frac{K}{\Lambda}$.

that the optimal linear one-shot DoF $K\gamma + N_0$ of the network can be achieved with a subpacketization as low as

$$S = \frac{K(K\gamma + N_0)}{(gcd(K, K\gamma, N_0))^2}, \tag{2.9}$$

where $gcd(\cdot)$ is the operator that computes the greatest common divisor. In equation (2.9), the term $gcd(K, K\gamma, N_0)$ represents the number of users that will store the same content in their caches. Therefore, in the delivery phase, this dedicated-cache setting can be seen as a shared-cache setting with $\Lambda = \frac{K}{gcd(K,K\gamma,N_0)}$ caches and $gcd(K, K\gamma, N_0)$ users per cache. We will refer to the scheme achieving the sum DoF of $K\gamma + N_0$ (for $N_0 \geq K\gamma$) with the subpacketization requirement in (2.9) as the *cyclic caching scheme*, where the term "cyclic" is chosen to reflect the structure of the aforementioned scheme.

Finite SNR regime: In addition to considering the DoF regime, we analyse the performance of the cyclic caching scheme in the finite signal-to-noise ratio (SNR) regime where we will consider MMSE-type optimized beamformers. In this regime, in line with other works on multi-antenna coded caching [47, 48], we will sacrifice some multiplexing gain $\alpha \leq N_0$ in order to have higher beamforming gains that compensate the well-known effects of the worst-user channel condition. At the same time, the multiplexing gain α can be also tuned to further reduce the subpacketization requirement of the scheme at the cost of a reduced sum DoF. Furthermore, while the optimized beamformer design problems tend to be non-convex in general and require computational complex iterative methods which can make the implementation infeasible even for a modest size of the network, we will see that the special unicasting nature of our transmissions makes the optimization problem quasi-convex, and hence, allows us to use uplink-downlink duality to attain a simple iterative solution. The low complexity of the beamforming design along with the low subpacketization result in a high-performance method that can be employed even when the number of users in the system is very large. This cyclic caching scheme is fully described and analysed in Chapter 6, and have been submitted for the following publication:

> [49] *M.J. Salehi, E. Parrinello, S.P. Shariatpanahi, P. Elia, A. Tölli, "Low-Complexity High-Performance Cyclic Caching for Large MISO Systems" under revision for publication to Transactions on Wireless Communications, 2021. The paper is available online at "https://arxiv.org/abs/2009.12231".*

2.3.3 Coded Caching with Heterogeneous Quality-of-Service Requirements

In many scenarios users download the requested content at different quality levels, depending on the type of device, type of subscription (e.g. base or premium), channel conditions and so on. That is why studying a caching model that captures the users' heterogeneity in terms of their required QoS is an important and pertinent problem. For the standard $K-$user shared-link MAN setting [2] discussed in Section 1.2.1, we now use a layered coding approach to encoded each file into H layers. While the first layer provides the video stream of base quality, the other subsequent layers can successively

refine this quality. When a user requests a file at QoS level $h \in [H]$, then this means that this user must be successfully delivered the first h layers (out of H) of the requested file. The number of users requesting QoS $h \in [H]$ is denoted by K_h and we refer to $\mathbf{K} = (K_1, K_2, \ldots, K_H)$ as the QoS profile. Moreover, assuming as always that each file has a normalized size of one unit, we will use r_h to denote the size of the first h layers of a file, which naturally yields that $r_H = 1$. For this setting, assuming that the vector $\mathbf{K}$ is known during caching, we propose a general placement and delivery scheme that achieves

$$T(t, \mathbf{K}) = \sum_{h=1}^{H} \sum_{g=0}^{K} \frac{\sum_{r=1}^{\mathfrak{P}(g,h)} \binom{K-r}{g}}{\binom{K}{g} N} x_{g,h} \tag{2.10}$$

where $\mathfrak{P}_{g,h} = \min\left\{K - g, K - \sum_{j=1}^{h-1} K_j\right\}$ and $\{x_{g,h}\}$ satisfies

$$\sum_{g=0}^{K} x_{g,h} = (r_h - r_{h-1})N, \quad h \in [H] \tag{2.11}$$

$$\sum_{g=0}^{K} g \cdot \left(\sum_{h=1}^{H} x_{g,h}\right) \leq tN, \tag{2.12}$$

$$x_{g,h} \geq 0, \quad h \in [H] \quad g \in [K]_0. \tag{2.13}$$

We will prove that by minimizing (2.10) with respect to $\{x_{g,h}\}$ and subject to the constraints in equations (2.11),(2.12), (2.13), we obtain the information theoretic optimal normalized delivery time under the assumptions that the cache placement is uncoded and aware of the QoS profile $\mathbf{K}$. An intriguing fact of this result is the ability of the developed tight converse bound to provide useful information on how to design the optimal caching scheme. A sub-optimal choice of $\{x_{g,h}\}$ which does not require knowledge of $\mathbf{K}$ during caching is also presented in our work. These results are fully described in Chapter 7 and resulted in the following publication

> [50] *E. Parrinello, A. Ünsal, and P. Elia, "Optimal coded caching under statistical QoS information," in 2019 IEEE International Symposium on Information Theory (ISIT),2019, pp. 2987–2991.*

2.3.4 Heterogeneous Coded Distributed Computing

In the context of distributed computing, one of the most well-known frameworks is the so-called *MapReduce* model which decomposes the computation in three distinct phases: the map phase, the shuffle phase and the reduce phase. As argued in [16] and references therein, the shuffle phase — which consists of an exchange of messages between the computing nodes — constitutes the major bottleneck that does not allow distributed computing to scale with the number of nodes. The work in [16] proposed a new variant of the MapReduce framework, called *Coded MapReduce*, which employs coded caching in the shuffling phase in order to reduce the inter-node communication load. For a MapReduce-type distributed computing system with Λ nodes that have to work on a

dataset of N files to obtain K output functions[4], Li et al. [16] showed that the optimal communication load, i.e. the minimum number of bits communicated by the nodes normalized by the total number of bits required by all the nodes, is given by

$$\tau^*_{CMR}(\gamma, \Lambda) = \frac{1}{\Lambda} \frac{\Lambda(1-\gamma)}{\Lambda\gamma}, \tag{2.14}$$

where γ is the fraction of the dataset that is stored by each computing node before the beginning of the map phase. The shuffling scheme that achieves this optimal load employs the same technique developed in [15] for the D2D cache-aided shared-link network. Motivated by the fact that it is common to have computing nodes with different storage and processing capabilities (as is commonly the case with Amazon EC2 clusters), in Chapter 8 we address the problem of heterogeneous distributed computing. We will assume that each computing node λ has computational power characterized by the constant c_λ ($0 \leq c_\lambda \leq 1$) and we denote by L_λ the number of output functions that are assigned to it. We will see that, for each computing node λ, an intuitive yet powerful approach is to select the value L_λ to be proportional to the computational power c_λ of the node, so that the time that the system spends in the reduce phase is lessened compared to when using a uniform assignment of the output functions among the nodes. With the vector $\mathbf{L} = (L_1, L_2, \ldots, L_\Lambda)$ at hand, assuming that we have the freedom to replicate the dataset t (referred to as the total *computation load*) times across the nodes, we will show that an achievable communication load is

$$\tau(t, \mathbf{L}) = \frac{\sum_{\lambda=1}^{\Lambda} L_\lambda(1-\gamma_\lambda)}{Kt}, \quad t \in [\Lambda], \tag{2.15}$$

where γ_λ has been defined in (2.2) and represents the fraction of the dataset that has to be stored in node λ. We will then prove that, for a fixed $\mathbf{L}$ and a fixed total computation load t, the above communication load is within a multiplicative gap of 2 from optimal under the assumptions of one-shot shuffling and homogeneous file assignment. A shuffling scheme is said "one-shot" if any node can recover each of its required data (intermediate values) from the intermediate values computed in the map phase and from at most one transmitted message by some other node. Furthermore, we say that a file assignment is homogeneous when all the files of the dataset are replicated among the nodes the same number of times. The achievable scheme and the technique used for the converse bound draw from those derived for the topology-aware shared-cache setting in Chapter 4, thus also showing the versatility of our developed techniques in this thesis. It is interesting to see that the allocation $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$ — where γ_λ was defined in equation (2.2) — of the total computation load t across the nodes as a function of $\mathbf{L}$, is crucial in allowing a shuffle phase that serves t computing nodes at a time. Furthermore, under the aforementioned assumption of homogeneous file assignment, we will prove that the expression in (2.15) serves as a lower bound for the optimal communication load of the heterogeneous coded distributed computing problem with fixed arbitrary output functions assignment $\mathbf{L}$ and

[4]The choice of using the symbol K for the total number of output functions is motivated by the fact that, as it will be clear in the corresponding chapter, the value of K is reminescent of the total number of requested files in a shared-cache setting with Λ caches.

fixed arbitrary per-node computation loads $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$ ($\gamma_\lambda \in \mathbb{R}$ and $0 \leq \gamma_\lambda \leq 1$), satisfying $\sum_{\lambda=1}^{\Lambda} \gamma_\lambda = t$. In conclusion, the proposed shuffle scheme exploits the heterogeneity of the output functions assignment $\mathbf{L}$ by properly allocating the total computation load t across the computing nodes, thus allowing for a reduced communication load compared to the standard coded MapReduce communication load in (2.14). These results are presented in Chapter 8 and will be also part of the following publication:

> [51] *E. Parrinello and P. Elia, "New optimality results for heterogeneous coded distributed computing," manuscript in preparation, 2021.*

Chapter 3

Topology-Agnostic Shared-Cache Networks

In this chapter we explore the fundamental limits of coded caching in the setting where a transmitter with potentially multiple (N_0) antennas serves different users that are assisted by a smaller number of caches. Under the assumption of uncoded cache placement and that the topology of the network is not known during this cache placement phase, we derive the exact optimal worst-case normalized delivery time and DoF, for a broad range of cache occupancy vectors where each such occupancy vector describes how many users are helped by each cache. This is achieved by presenting an information-theoretic converse based on index coding that succinctly captures the impact of the cache occupancy vector, as well as by presenting a coded caching scheme that optimally adapts to the heterogeneity of the topology by exploiting the benefits of encoding across users that share the same cache.

3.1 System Model and Problem Formulation

We consider a shared-link configuration with a transmitting server having N_0 transmitting antennas and access to a library of N files $W^{(1)}, W^{(2)}, \dots, W^{(N)}$, each of size equal to one unit of 'file', where this transmitter is connected to K receiving users and to $\Lambda \leq K$ helper nodes that will serve as caches which store content from the library. The communication process is split into the cache placement phase, the user-to-cache assignment phase, and the delivery phase.

Cache placement phase During this phase, helper nodes (caches) store content from the library without having knowledge of the users' requests. Each helper cache has size $M \leq N$ ($\gamma \triangleq \frac{M}{N}$) units of file, and no coding is applied to the content stored at the helper caches; this corresponds to the common case of *uncoded cache placement*. We will denote by $\mathcal{Z}_\lambda$ the content stored by helper node λ during this phase such that the vector $\mathbf{Z} = (\mathcal{Z}_1, \mathcal{Z}_2, \dots, \mathcal{Z}_\Lambda)$ represents the overall cache placement. We assume that the cache placement algorithm is oblivious of the subsequent user-to-cache association, i.e. during

the cache placement phase we have no knowledge of the number of users that will be assisted by each cache. In other words, during the cache placement phase the topology of the network is not known yet. This condition will be removed in the next chapter where we will assume that we can allocate the memory to each cache as a function of the topology.

User-to-cache association phase After the caches are filled, each user is assigned to exactly *one* helper node/cache, from which it can download content at zero cost. Specifically, each cache $\lambda = 1, 2, \ldots, \Lambda$, is assigned to a set of users $\mathcal{U}_\lambda$, and all these disjoint sets

$$\mathcal{U} \triangleq \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_\Lambda\}$$

form a partition of the set of users $\{1, 2, \ldots, K\}$, describing the overall association of the users to the caches. This cache assignment is independent of the cached content and independent of the file requests to follow. We here consider any arbitrary user-to-cache association $\mathcal{U}$, thus allowing the results to reflect both an ability to choose/design the association, as well as to reflect possible association restrictions due to randomness or topology. Similarly, having the user-to-cache association being independent of the requested files, is meant to reflect the fact that such associations may not be able to vary as quickly as a user changes the requested content.

Delivery phase The delivery phase starts when each user $k = 1, \ldots, K$ requests from the transmitter, any *one* file $W^{(d_k)}$, $d_k \in \{1, \ldots, N\}$ out of the N files of the library. Upon notification of the entire *demand vector* $\mathbf{d} = (d_1, d_2, \ldots, d_K) \in [N]^K$, the transmitter aims to deliver the requested files and the objective is to design a caching scheme and a delivery scheme that do so with limited (delivery phase) duration T, where the delivery algorithm has full knowledge of the user-to-cache association $\mathcal{U}$. For each transmission, the received signal at user k takes the form

$$y_k = \boldsymbol{h}_k^T \boldsymbol{x} + w_k, \quad k = 1, \ldots, K \tag{3.1}$$

where $\boldsymbol{x} \in \mathbb{C}^{N_0 \times 1}$ denotes the transmitted vector satisfying a power constraint $\mathbb{E}(||\boldsymbol{x}||^2) \leq P_T$, $\boldsymbol{h}_k \in \mathbb{C}^{N_0 \times 1}$ denotes the channel gain of receiver k, and w_k represents the additive white gaussian noise (AWGN) noise with unit power at user k. We will assume that the maximum allowed average power P_T is high (i.e., we will assume high signal-to-noise ratio (SNR)), that there exists perfect channel state information throughout the active users, that fading is statistically symmetric, and that each link (one antenna to one receiver) has ergodic capacity $\log(SNR) + o(log(SNR))$.

Observation 1. *We notice that in the high-SNR regime of interest, when $N_0 = 1$ and $\Lambda = K$ (where each cache is associated to one user), the setting matches identically the original single-stream setting in [2]. In particular, the file size and log(SNR) are here scaled so that, as in [2], each point-to-point link has capacity of 1 file per unit of time. When $N_0 > 1$ and $\Lambda = K$ (again each cache is associated to one user), the setting matches the multi-server setting of [7], which we now explore in the presence of fewer caches serving potentially different numbers of users.*

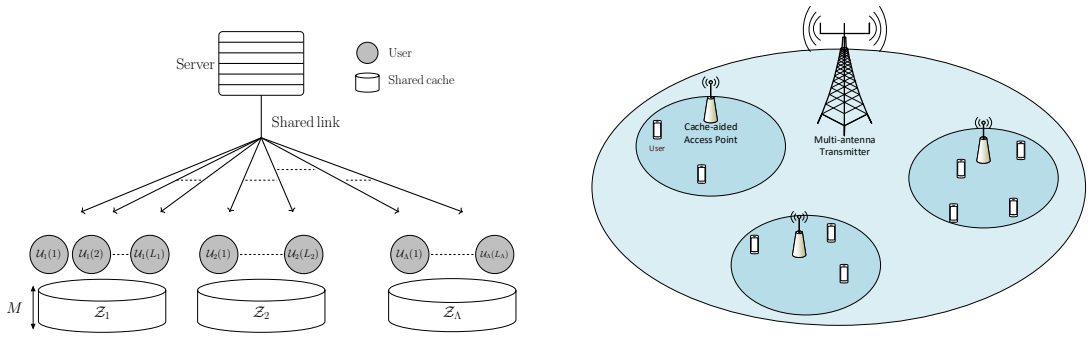


Figure 3.1 – Schematic of the shared-link BC with shared caches (left) and a pictorial representation of the multi-antenna BC with shared caches (right).

Performance measure As one can imagine, some user-to-cache associations $\mathcal{U}$ may allow for higher performance than others; for instance, considering the fact that $\mathcal{U}$ is not known during the placement phase, one can suspect that a uniform distribution of the users among the caches may be preferable. Part of the objective of this work is to explore the effect of such associations on the overall performance. Toward this, for any given $\mathcal{U}$, we define cache occupancy vector the sorted tuple

$$\mathbf{L} = (L_1, \ldots, L_\Lambda)$$

where L_λ is the number of users assigned to the λ-th *most populated* helper node/cache[1]. Naturally, $\sum_{\lambda=1}^{\Lambda} L_\lambda = K$. Each vector $\mathbf{L}$ defines a *class* $\mathcal{U}_\mathbf{L}$ comprising all the user-to-cache associations $\mathcal{U}$ that correspond to the same cache occupancy vector $\mathbf{L}$.

As in [2], the measure of interest T is the number of time slots needed to complete delivery of any request vector[2] $\mathbf{d}$. For a given fixed $\mathbf{Z}$, we use $T^*(\mathcal{U}, \mathbf{d}, \mathbf{Z})$ to define the optimal normalized delivery time required to satisfy demand $\mathbf{d}$ in the presence of a user-to-cache association described by $\mathcal{U}$. The optimal worst-case delivery time for a given cache occupancy vector $\mathbf{L}$ can be formally stated as

$$T^*(\mathbf{L}) \triangleq \min_{\mathbf{Z}} \max_{(\mathcal{U},\mathbf{d})\in(\mathcal{U}_\mathbf{L},\{1,\ldots,N\}^K)} T^*(\mathcal{U}, \mathbf{d}, \mathbf{Z}). \tag{3.2}$$

Our interest is in the regime of $N \geq K$ where there are more files than users.

3.2 Main Results

In this section we present the main results for the considered topology-agnostic shared-cache problem. We first describe the main results for the single-antenna case (shared-link BC), and then generalize to the multi-antenna case.

[1]Here $\mathbf{L}$ is simply the vector of the cardinalities of $\mathcal{U}_\lambda$, $\forall \lambda \in \{1, \ldots, \Lambda\}$, sorted in descending order. For example, $L_1 = 6$ states that the highest number of users served by a single cache, is 6.

[2]The time scale is normalized such that one time slot 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.

3.2.1 Topology-Agnostic Shared-Link Coded Caching with Shared Caches

The following theorem presents the main result for the shared-link case ($N_0 = 1$).

Theorem 1. *In the K-user shared-link channel with Λ shared caches of normalized size γ, the topology-agnostic optimal delivery time for any cache occupancy vector $\mathbf{L}$ is*

$$T^*(\mathbf{L}) = \mathrm{Conv}_{t\in[\Lambda]_0}\left(\frac{\sum_{r=1}^{\Lambda-t} L_r\binom{\Lambda-r}{t}}{\binom{\Lambda}{t}}\right), \tag{3.3}$$

where $t = \Lambda\gamma$.

Proof. The achievability part of the proof is given in Section 3.3, and the converse is proved in Section 3.4 after setting $N_0 = 1$.

Remark 1. *We note that the converse that supports Theorem 1, encompasses the class of all caching and delivery schemes that employ uncoded cache placement under a general sum cache constraint $\frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}|\mathcal{Z}_\lambda| = M$ which does not* necessarily *impose an individual cache size constraint. The converse also encompasses all scenarios that involve a library of size $\sum_{n\in[N]}|W^{(n)}| = N$ but where the files may be of different size. In the end, even though the designed optimal scheme will consider an individual cache size M and equal file sizes, the converse guarantees that — if the cache occupancy vector $\mathbf{L}$ is not known during caching — there cannot exist a scheme (even in settings with uneven cache sizes or uneven file sizes) that exceeds the optimal performance identified here .*

From Theorem 1, we see that in the uniform case[3] where $\mathbf{L}_{unif} = (\frac{K}{\Lambda}, \frac{K}{\Lambda}, \ldots, \frac{K}{\Lambda})$, the expression in (3.3) reduces to

$$T^*(\mathbf{L}_{unif}) = \frac{K(1-\gamma)}{\Lambda\gamma + 1},$$

matching the achievable delay presented in [6] for the shared-cache setting with uniform $\mathbf{L}$. It also matches the result by [30] which proved that this performance — in the context of the multiple file request problem — is optimal under the assumption of uncoded cache placement. The following corollary relates to this uniform case.

Corollary 1. *In the uniform case where $\mathbf{L}_{unif} = (\frac{K}{\Lambda}, \frac{K}{\Lambda}, \ldots, \frac{K}{\Lambda})$, the aforementioned optimal normalized delivery time $T^*(\mathbf{L}_{unif})$ is smaller than the corresponding delay $T^*(\mathbf{L})$ for any other non-uniform cache occupancy vector $\mathbf{L}$.*

Proof. The proof that the uniform cache occupancy vector results in the smallest delay, follows directly from the fact that in (3.3), both L_r and $\binom{\Lambda-r}{\Lambda\gamma}$ are non-increasing with r. □

Remark 2 (Shared-link coded caching with multiple file requests)**.** *In the error-free shared-link case ($N_0 = 1$), with worst-case demand assumptions, i.e., when each user*

[3]Here, this uniform case, naturally implies that $\Lambda|K$.

requests a different file, the shared-cache problem here is closely related to the coded caching problem with multiple file requests per user, where now Λ users with their own cache, request in total $K \geq \Lambda$ different files. The demand vector $\mathbf{d}$ would now represent the vector of the indices of the K requested files and for each user $\lambda \in [\Lambda]$, $\mathcal{U}_\lambda$ would tell us that user λ requests files $\{W^{(d_k)}, k \in \mathcal{U}_\lambda\}$. At this point, as before, the problem is now defined by the user-to-file association $\mathcal{U} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_\Lambda\}$ which describes — given a fixed demand vector $\mathbf{d}$ — the files requested by any user. From this point on, the equivalence with the original shared cache problem is complete. As before, each such $\mathcal{U}$ again has a corresponding (sorted) cache occupancy vector $\mathbf{L} = (L_1, L_2, \ldots, L_\Lambda)$, and belongs to a class $\mathcal{U}_\mathbf{L}$ with all other associations $\mathcal{U}$ that share the same occupancy vector $\mathbf{L}$. As we quickly show in the Appendix Section A.3, our scheme and converse can be adapted to the multiple file request problem, and thus directly from Theorem 1 we conclude that for this multiple file request problem, the optimal delay corresponding to any cache occupancy vector $\mathbf{L}$, takes the form $T^*(\mathbf{L}) = \mathrm{Conv}_{\Lambda\gamma \in [\Lambda]_0} \left(\frac{\sum_{r=1}^{\Lambda - \Lambda\gamma} L_r \binom{\Lambda - r}{\Lambda\gamma}}{\binom{\Lambda}{\Lambda\gamma}} \right)$. At this point we close the parenthesis regarding multiple file requests, and we refocus exclusively on the problem of shared caches.

3.2.2 Topology-Agnostic Multi-Antenna Coded Caching with Shared Caches

The following theorem extends Theorem 1 to the case where the transmitter is equipped with multiple ($N_0 > 1$) antennas.

Theorem 2. *In the N_0-antenna K-user broadcast channel with Λ shared caches of normalized size γ, the optimal delivery time for any cache occupancy vector $\mathbf{L}$ satisfying $L_\lambda \geq N_0, \ \forall \lambda \in [\Lambda]$ is*

$$T^*(\mathbf{L}, N_0) = \frac{1}{N_0} \mathrm{Conv}_{t \in [\Lambda]_0} \left(\frac{\sum_{r=1}^{\Lambda - t} L_r \binom{\Lambda - r}{t}}{\binom{\Lambda}{t}} \right), \tag{3.4}$$

where $t = \Lambda\gamma$. This reveals a multiplicative gain of N_0 with respect to the single antenna case.

Proof. The scheme that achieves (3.4) is presented in Section 3.3, and the converse is presented in Section 3.4.

Theorem 3. *In the uniform case of $\mathbf{L}_{unif} = \left(\frac{K}{\Lambda}, \frac{K}{\Lambda}, \ldots, \frac{K}{\Lambda}\right)$, the following normalized delivery time is achievable*

$$T^*(\mathbf{L}_{unif}, N_0) = \begin{cases} \dfrac{K - K\gamma}{N_0(\Lambda\gamma + 1)} & \Lambda < \frac{K}{N_0}, \quad (3.5a) \\ \dfrac{K - K\gamma}{N_0(\frac{K}{N_0}\gamma + 1)} = \dfrac{K - K\gamma}{K\gamma + N_0} & \Lambda \geq \frac{K}{N_0}. \quad (3.5b) \end{cases}$$

The performance for the regime $\Lambda < \frac{K}{N_0}$ is optimal under the assumption of uncoded cache placement, while for the opposite regime where $\Lambda \geq \frac{K}{N_0}$ optimality is proven among the linear one-shot schemes. Furthermore, the achievability of (3.5b) *requires that N_0 is an integer multiple of $\frac{K}{\Lambda}$.*

	Dedicated Caches	
	Single Antenna	Multiple Antennas
Achievable delay	$\frac{K(1-\gamma)}{K\gamma+1}$ [2]	$\frac{K(1-\gamma)}{K\gamma+N_0}$ [7]
Optimality	gap of 2 [3] and optimality under uncoded prefetching [5, 52]	optimal under one-shot linear schemes [22]

	Shared Caches/ Multiple File Requests		Shared Caches	
	Single Antenna		Multiple Antennas	
Cache occupancy	uniform	not uniform	uniform	not uniform ($L_\Lambda \geq N_0$)
Achievable delay	$\frac{K(1-\gamma)}{\Lambda\gamma+1}$ [30]	$\frac{\sum_{r=1}^{\Lambda-\Lambda\gamma} L_r \binom{\Lambda-r}{\Lambda\gamma}}{\binom{\Lambda}{\Lambda\gamma}}$	(1) $\frac{K-K\gamma}{N_0(\Lambda\gamma+1)}$ $\Lambda < \frac{K}{N_0}$ (2) $\frac{K-K\gamma}{K\gamma+N_0}$ $\Lambda \geq \frac{K}{N_0}$	$\frac{\sum_{r=1}^{\Lambda-\Lambda\gamma} L_r \binom{\Lambda-r}{\Lambda\gamma}}{N_0\binom{\Lambda}{\Lambda\gamma}}$
Optimality	under uncoded prefetching [30]	under uncoded prefetching	(1) under uncoded prefetching (2) under one-shot linear schemes [22]	under uncoded prefetching

Table 3.1 – Overview of the optimal topology-agnostic worst-case performance for the cache-aided networks with dedicated and shared caches.

Proof. The result in (3.5a) for the regime where $\Lambda < \frac{K}{N_0}$, follows directly from Theorem 2. The optimality under the one-shot linear assumption of the delivery time in (3.5b) has been proven in [22] for the cache-aided MISO setting, while the scheme achieving such a performance with a limited number of caches $\Lambda \geq \frac{K}{N_0}$ is presented in Section 3.5. □

Before proceeding with the interpretation of the results, we point out that in Chapter 6 we will present another scheme that achieves the same performance as (3.5b) for the specific regime where $N_0 \geq K\gamma$. The scheme described in Chapter 6 has the advantage of having a very low subpacketization requirement, which we will compare with other known one-shot linear sum-DoF optimal schemes.

3.2.3 Interpretation of Results

Capturing the effect of the cache occupancy vector

In a nutshell, Theorems 1,2 quantify how non-uniformities in the vectors $\mathbf{L}$ bring about increased delays. What we see is that, the more skewed the cache occupancy vector is, the larger is the delay. This is reflected in Figure 3.2 which shows — for a setting with $K = 30$ users and $\Lambda = 6$ caches — the memory-delay trade-off curves for different cache occupancy vectors $\mathbf{L}$. As expected, Figure 3.2 demonstrates that when all users

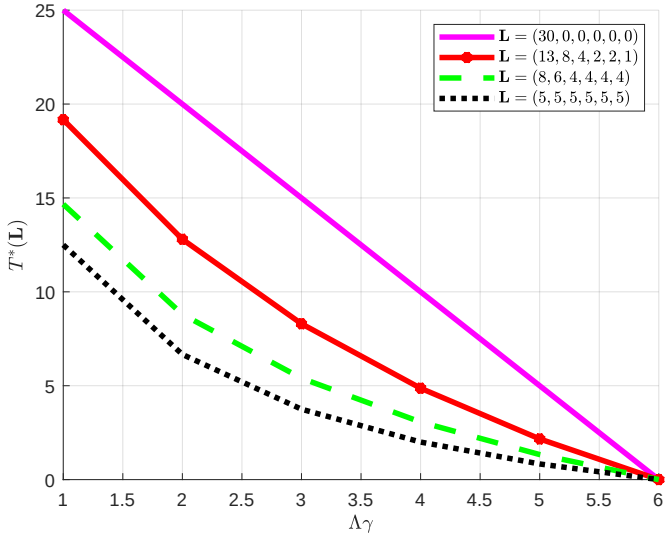


Figure 3.2 – Optimal delivery time for different cache occupancy vectors $\mathbf{L}$, for $K = 30$ users and $\Lambda = 6$ caches.

are connected to the same helper cache, the only gain arising from caching is the well known *local caching gain*. On the other hand, when users are assigned uniformly among the caches (i.e., when $L_\lambda = \frac{K}{\Lambda}, \forall \lambda \in [\Lambda]$) the caching gain is maximized and the delay is minimized.

A multiplicative reduction in delay

Theorem 2 states that, as long as each cache is associated to at least N_0 users, we can achieve a normalized delivery time $T^*(\mathbf{L}, N_0) = \frac{1}{N_0} \frac{\sum_{r=1}^{\Lambda-\Lambda\gamma} L_r \binom{\Lambda-r}{\Lambda\gamma}}{\binom{\Lambda}{\Lambda\gamma}}$. The resulting reduction

$$\frac{T(\mathbf{L}, N_0 = 1)}{T(\mathbf{L}, N_0)} = N_0 \tag{3.6}$$

as compared to the single-stream case, comes in strong contrast to the case of $\Lambda = K$ where, as we know from [7], this same reduction takes the form

$$\frac{T(\Lambda = K, N_0 = 1)}{T(\Lambda = K, N_0)} = \frac{\frac{K(1-\gamma)}{1+\Lambda\gamma}}{\frac{K(1-\gamma)}{N_0+\Lambda\gamma}} = \frac{N_0 + \Lambda\gamma}{1 + \Lambda\gamma} \tag{3.7}$$

which approaches N_0 only when $\gamma \to 0$, and which decreases as γ increases.

In the uniform case ($L_\lambda = \frac{K}{\Lambda}$) with $\Lambda \leq \frac{K}{N_0}$, Theorem 3 implies a sum-DoF

$$\frac{K(1-\gamma)}{T} = N_0(1 + \Lambda\gamma)$$

which reveals that every time we increase $\Lambda\gamma$ by one we gain N_0 degrees of freedom. This is in direct contrast to the case of $\Lambda = K$ (for which case we recall from [7] that the DoF

is $N_0 + \Lambda\gamma = N_0 + K\gamma$) where the same unit increase in the cache redundancy yields only one additional DoF.

Impact of encoding over users that share the same cache

As we know, both the MAN algorithm in [2] and the multi-antenna algorithm in [7], are designed for users with different caches, so — in the uniform case where $L_\lambda = K/\Lambda$ — one conceivable treatment of the shared-cache problem would have been to apply these algorithms over Λ users at a time, all with different caches[4]. As we see, in the single antenna case, this implementation would treat $1 + \Lambda\gamma$ users at a time thus yielding a delay of $T = \frac{K(1-\gamma)}{1+\Lambda\gamma}$, while in the multi-antenna case, this implementation would treat $N_0 + \Lambda\gamma$ users at a time (see [7]) thus yielding a delay of $T = \frac{K(1-\gamma)}{N_0+\Lambda\gamma}$. What we see here is that while this direct implementation is optimal in the single antenna case (see also [30]), in the multi-antenna case this same approach can have a performance gap of

$$\begin{cases} \frac{\frac{K(1-\gamma)}{N_0+\Lambda\gamma}}{\frac{K(1-\gamma)}{N_0(1+\Lambda\gamma)}} = \frac{N_0(1+\Lambda\gamma)}{N_0+\Lambda\gamma} & \Lambda < \frac{K}{N_0} \\ \frac{\frac{K(1-\gamma)}{N_0+\Lambda\gamma}}{\frac{K(1-\gamma)}{N_0+K\gamma}} = \frac{N_0(1+\Lambda\gamma)}{N_0+K\gamma} & \Lambda \geq \frac{K}{N_0} \end{cases} \tag{3.8}$$

from the derived optimal performance in (3.5a).

Interpretations of the above results in the dedicated-cache setting

As we have discussed already in the second chapter, the shared-cache setting can also reflect the limits imposed by the subpacketization bottleneck of coded caching in the dedicated-cache setting. The work in [32] proposed a decentralized framework for the subpacketization-constrained cache-aided shared-link problem, where only Λ cache contents are designed such that during the cache placement phase each of the users in the system picks at random any one of these designed Λ caches. The choice of limiting Λ to be smaller than K is taken to reduce the subpacketization requirement, which is now limited to $S = \binom{\Lambda}{\Lambda\gamma}$, and which we recall that represents the number of subfiles in which we have to subpacketize each file of the library. Because of this decentralized placement, the topology of the network in the delivery phase can be represented by the cache occupancy vector $\mathbf{L} = (L_1, \ldots, L_\Lambda)$, where L_λ represents the number of users that have stored the designed cache indexed by λ. It can be verified that their delivery phase is isomorphic to the delivery phase of our shared-cache setting. At the same time, their proposed scheme coincides with the single-antenna version of our developed scheme. This one-to-one mapping between these two settings implies that the optimality of the performance in Theorem 1 applies also to the decentralized dedicated-cache setting in [32]. Furthermore, in a centralized dedicated-cache scenario where we can control the cache placement at the users such that $\mathbf{L} = (\frac{K}{\Lambda}, \ldots, \frac{K}{\Lambda})$, the result in Theorem 3 tells us that it is sufficient to design $\Lambda = \frac{K}{N_0}$, as it was done originally in [33], to achieve the full linear

[4]This would then require $\frac{K}{\Lambda}$ such rounds in order to cover all K users.

sum-Dof $K\gamma + N_0$ of the network, and increasing this value of Λ cannot help in boosting the performance.

3.3 General Achievable Scheme for $N_0 \leq L_\Lambda$

This section is dedicated to the description of the placement and delivery schemes achieving the performance presented in Theorem 1, Theorem 2, and in Theorem 3 for the regime where $\Lambda \leq \frac{K}{N_0}$. The formal description of the optimal scheme in the upcoming subsection will be followed by a clarifying example in Section 3.3.4 that demonstrate the main idea behind the design.

3.3.1 Description of the Scheme

The placement phase, which uses exactly the algorithm developed in [2] for the case of $(\Lambda = K, M, N)$, is independent of $\mathcal{U}, \mathbf{L}$, while the delivery phase is designed for any given $\mathcal{U}$, and will achieve the optimal worst-case delivery times stated in (3.3) and (3.4). As mentioned already, we will assume that any non zero L_λ satisfies $L_\lambda \geq N_0, \forall \lambda \in [\Lambda]$.

Cache Placement Phase

The placement phase employs the original cache-placement algorithm of [2] corresponding to the scenario of having only Λ users, each with their own cache. Hence — recalling from [2] — first each file $W^{(n)}$ is split into $\binom{\Lambda}{\Lambda\gamma}$ disjoint subfiles $W^{(n)}_{\mathcal{T}}$, for each $\mathcal{T} \subset [\Lambda]$, $|\mathcal{T}| = \Lambda\gamma$, and then each cache stores a fraction γ of each file, as follows

$$\mathcal{Z}_\lambda = \{W^{(n)}_{\mathcal{T}} : \mathcal{T} \ni \lambda,\ \forall n \in [N]\}. \tag{3.9}$$

Delivery Phase

For the purpose of the scheme description only, we will assume without loss of generality that $|\mathcal{U}_1| \geq |\mathcal{U}_2| \geq \cdots \geq |\mathcal{U}_\Lambda|$ (any other case can be handled by simple relabeling of the caches), so that $L_\lambda = |\mathcal{U}_\lambda|$. Furthermore, in a slight abuse of notation, we will consider here each $\mathcal{U}_\lambda$ to be an *ordered vector* describing, in order, the users associated to cache λ. We will also use

$$\boldsymbol{s_\lambda} = (\mathcal{U}_\lambda \| \mathcal{U}_\lambda)_{N_0}, \lambda \in [\Lambda] \tag{3.10}$$

to denote the N_0-fold concatenation of each $\mathcal{U}_\lambda$. Each such $N_0 L_\lambda$-length vector $\boldsymbol{s_\lambda}$ can be seen as the concatenation of L_λ different N_0-tuples $\boldsymbol{s_{\lambda,j}}$, $j = 1, 2, \ldots, L_\lambda$, i.e., each $\boldsymbol{s_\lambda}$ takes the form[5]

$$\boldsymbol{s_\lambda} = \boldsymbol{s_{\lambda,1}} \| \boldsymbol{s_{\lambda,2}} \| \ldots \| \underbrace{\boldsymbol{s_{\lambda,L_\Lambda}}}_{N_0-\text{length}}.$$

[5]Note also that having $L_\lambda \geq N_0, \forall \lambda \in [\Lambda]$ guarantees that in any given $\boldsymbol{s_{\lambda,j}}, j \in [L_\lambda]$, a user appears at most once.

The delivery phase commences with the demand vector $\mathbf{d}$ being revealed to the server. Delivery will consist of L_1 rounds, where each round $j \in [L_1]$ serves users

$$\mathcal{R}_j = \bigcup_{\lambda \in [\Lambda]} \big(\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}} : L_\lambda \geq j\big). \tag{3.11}$$

Transmission scheme Once the demand vector $\mathbf{d}$ is revealed to the transmitter, each requested subfile $W_{\mathcal{T}}^{(n)}$ (for any n found in $\mathbf{d}$) is further split into N_0 mini-files $\{W_{\mathcal{T},l}^{(n)}\}_{l\in[N_0]}$. During round j, serving users in $\mathcal{R}_j$, we create $\binom{\Lambda}{\Lambda\gamma+1}$ sets $\mathcal{Q} \subseteq [\Lambda]$ of size $|\mathcal{Q}| = \Lambda\gamma + 1$, and for each set $\mathcal{Q}$, we pick the set of users

$$\chi_{\mathcal{Q}} = \bigcup_{\lambda \in \mathcal{Q}} \big(\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}} : L_\lambda \geq j\big). \tag{3.12}$$

If $\chi_{\mathcal{Q}} = \emptyset$, then there is no transmission, and we move to the next $\mathcal{Q}$. If $\chi_{\mathcal{Q}} \neq \emptyset$, the server — during this round j — transmits the following vector[6]

$$\boldsymbol{x}_{\chi_{\mathcal{Q}}} = \sum_{\lambda \in \mathcal{Q}: L_\lambda \geq j} \mathbf{H}_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}}^{-1} \times \begin{bmatrix} W_{\mathcal{Q}\setminus\{\lambda\},l_1}^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(1)})} \\ \vdots \\ W_{\mathcal{Q}\setminus\{\lambda\},l_k}^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)})} \\ \vdots \\ W_{\mathcal{Q}\setminus\{\lambda\},l_{N_0}}^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(N_0)})} \end{bmatrix} \tag{3.13}$$

where $W_{\mathcal{Q}\setminus\{\lambda\},l_k}^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)})}$ is a mini-file intended for user $\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)$, i.e., for the user labelled by the kth entry of vector $\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}$. The choice of each $l_k \in [N_0]$ is sequential, guaranteeing that no mini-file $W_{\mathcal{Q}\setminus\{\lambda\},l_k}^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)})}$ of the same subfile $W_{\mathcal{Q}\setminus\{\lambda\}}^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)})}$ is transmitted twice. In the above, $\mathbf{H}_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}}^{-1}$ denotes the inverse of the channel matrix between the N_0 transmit antennas and the users in vector $\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}$.

Since each user appears in $\boldsymbol{s}_{\boldsymbol{\lambda}}$ (and consequently in $\bigcup_{j\in[\mathcal{L}_1]} \mathcal{R}_j$) exactly N_0 times, at the end of the L_1 rounds, all the N_0 mini-files $W_{\mathcal{Q}\setminus\{\lambda\},l_k}^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)})}$, $l_k \in [N_0]$ will be sent once. This, along with the fact that the sets $\mathcal{Q}$ follow the standard MAN scheme in [2], guarantees that the scheme serves all subfiles requested by the users.

Decoding Directly from (3.13), we see that each receiver $\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)$ obtains a received signal whose noiseless version takes the form

$$y_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)} = W_{\mathcal{Q}\setminus\{\lambda\},l_k}^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)})} + \iota_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)}$$

[6]The transmitted-vector structure below draws from the structure in [33], in the sense that it involves the linear combination of one or more *Zero Forcing* precoded (ZF-precoded) vectors of subfiles that are labeled (as we see below) in the spirit of [2].

where $\iota_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)}$ is the k-th entry of the interference vector

$$\sum_{\substack{\lambda' \in \mathcal{Q}\setminus\{\lambda\}: \\ L_{\lambda'} \geq j}} \mathbf{H}^{-1}_{\boldsymbol{s}_{\boldsymbol{\lambda}',\boldsymbol{j}}} \cdot \left[W^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda}',\boldsymbol{j}}(1)})}_{\mathcal{Q}\setminus\{\lambda'\},l_1} \cdots W^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda}',\boldsymbol{j}}(k)})}_{\mathcal{Q}\setminus\{\lambda'\},l_k} \cdots W^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda}',\boldsymbol{j}}(N_0)})}_{\mathcal{Q}\setminus\{\lambda'\},l_{N_0}} \right]^T. \tag{3.14}$$

In the above, we see that the entire interference term $\iota_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)}$ experienced by receiver $\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}(k)$, can be removed (cached-out) because all appearing subfiles

$$W^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda}',\boldsymbol{j}}(1)})}_{\mathcal{Q}\setminus\{\lambda'\},l_1}, \ldots, W^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda}',\boldsymbol{j}}(N_0)})}_{\mathcal{Q}\setminus\{\lambda'\},l_{N_0}}$$

, for all $\lambda' \in \mathcal{Q} \setminus \{\lambda\}, L_{\lambda'} \geq j$, can be found in cache λ associated to this user, simply because $\lambda \in \mathcal{Q}\setminus\{\lambda'\}$.

This completes the proof of the scheme for the multi-antenna case.

Small modification for the single antenna case

For the single-antenna case, the only difference is that now $\boldsymbol{s}_{\boldsymbol{\lambda}} = \mathcal{U}_\lambda$, and that each transmitted vector in (3.13) during round j, becomes a scalar of the form

$$x_{\chi_{\mathcal{Q}}} = \bigoplus_{\lambda \in \mathcal{Q}: L_\lambda \geq j} W^{(d_{\boldsymbol{s}_{\boldsymbol{\lambda},\boldsymbol{j}}})}_{\mathcal{Q}\setminus\{\lambda\},1}. \tag{3.15}$$

The rest of the details from the general scheme, as well as the subsequent calculation of the delay, follow directly. We also notice that this same scheme for the single antenna case has been already proposed for the previously discussed decentralized coded caching setting with reduced subpacketization in [32, 53].

3.3.2 Calculation of the Normalized Delivery Time

To first calculate the delay needed to serve the users in $\mathcal{R}_j$ during round j, we recall that there are $\binom{\Lambda}{\Lambda\gamma+1}$ sets

$$\chi_{\mathcal{Q}} = \bigcup_{\lambda \in \mathcal{Q}} \big(\mathcal{U}_\lambda(j) : L_\lambda \geq j\big), \mathcal{Q} \subseteq [\Lambda]$$

of users, and we recall that $|\mathcal{U}_1| \geq |\mathcal{U}_2| \geq \cdots \geq |\mathcal{U}_\Lambda|$. For each such non-empty set, there is a transmission. Furthermore we see that for $a_j \triangleq \Lambda - \frac{|\mathcal{R}_j|}{N_0}$, there are $\binom{a_j}{\Lambda\gamma+1}$ such sets $\chi_{\mathcal{Q}}$ which are empty, which means that round j consists of

$$\binom{\Lambda}{\Lambda\gamma+1} - \binom{a_j}{\Lambda\gamma+1} \tag{3.16}$$

transmissions.

Since each file is split into $\binom{\Lambda}{\Lambda\gamma} N_0$ subfiles, the duration of each such transmission is

$$\frac{1}{\binom{\Lambda}{\Lambda\gamma} N_0} \tag{3.17}$$

and thus summing over all L_1 rounds, the total delay takes the form

$$T = \frac{\sum_{j=1}^{L_1} \binom{\Lambda}{\Lambda\gamma+1} - \binom{a_j}{\Lambda\gamma+1}}{\binom{\Lambda}{\Lambda\gamma} N_0} \tag{3.18}$$

which, after some basic algebraic manipulation (see Appendix A.2.7 for the details), takes the final form

$$T = \frac{1}{N_0} \frac{\sum_{r=1}^{\Lambda-\Lambda\gamma} L_r \binom{\Lambda-r}{\Lambda\gamma}}{\binom{\Lambda}{\Lambda\gamma}} \tag{3.19}$$

which concludes the achievability part of the proof. □

3.3.3 Intuition on the Scheme: Separability and the Parallel Version of the Multi-Antenna BC with Shared Caches

The scheme described above makes use of a *non-separable* approach that is based on multicast messages that simultaneously exploit the caches and the channel. To gain some insight into this scheme, we will briefly look at a simpler (but separable) solution to the easier parallel version of the multi-transmitter BC with shared caches.

N_0-transmitter parallel BC with shared caches This parallel version of the problem consists of N_0 transmitters, each connected to all K receivers via a broadcast link of normalized unit capacity. The main difference in this parallel version is that naturally each receiver enjoys N_0 simultaneous signal observations, one from each transmitter. As in our setting, each receiver has access to one of Λ caches.

In this parallel version, an optimal scheme would consist of a cache placement that is identical to ours, and a delivery phase which begins by generating the same set of XORs that we would generate for the single-antenna case, then partitioning this set into any N_0 equal-sized subsets[7], and finally the delivery phase would conclude by sending each of the N_0 subsets, simultaneously, from each of the transmitters. Decoding is done exactly as in the single-antenna case. This achieves a speed-up factor of N_0 over the single-antenna case, for any cache occupancy vector $\mathbf{L}$ (including for the case where $L_\lambda \leq N_0$). As it will become clear later, the lower bound in Section 3.4 applies also to this parallel channel, and we can thus conclude that the optimal performance for the parallel channel coincides with the derived performance for the MISO BC for $L_\lambda \geq N_0$.

Non-separability of the scheme for the MISO-BC with shared caches To draw the connection to our scheme, we first note that the above described *parallel* version is separable; the scheme consists of a) a cache placement and multicast message generation that are independent of the network/channel and are identical to the single-transmitter case, and b) an efficient delivery (now as a function of the network) of these multicast messages. However, this is not the case for the MISO BC setting of interest here, where

[7] Without loss of generality, we here assume that the number of XORs is a multiple of the number of antennas. The general case can be handled with a small increase in the subpacketization.

the multicast messages themselves are a function of the channel. The difference here is that, to the best of our understanding, we could not have used the single-stream XORs generated in Section 3.3.1, and then simply found a way (e.g. via beamforming) to deliver these over a parallelized channel, where this channel was *parallelized* (for example by the use of Zero Forcing)[8]. Instead, what we do is that we consider the same sequence of XORs as in the parallel channel setting, we split these into the same N_0 subsets, but then instead of considering XORs, we decompose each XOR, and linearly combine the subfiles within each such XOR, as a function of the channel, in a way that the created multicast messages — in addition to exploiting the caches of the users addressed by each such multicast message — also simultaneously orthogonalize the channels to another set of users that share the same cache. This approach makes the scheme non-separable. Whether it is possible or not to develop a fully separable optimal solution for the MISO BC with shared caches, remains an open question.

3.3.4 Illustrative Example

Consider a scenario with $K = 15$ users $\{1, 2, \ldots, 15\}$, a server equipped with $N_0 = 2$ transmitting antennas that stores a library of $N = 15$ equally-sized files $W^{(1)}, W^{(2)}, \ldots, W^{(15)}$, and consider $\Lambda = 3$ helper caches, each of size equal to $M = 5$ units of file.

In the cache placement phase, we split each file $W^{(n)}$ into 3 equally-sized disjoint subfiles denoted by $W_1^{(n)}, W_2^{(n)}, W_3^{(n)}$ and as in [2], each cache λ stores $W_\lambda^{(n)}, \forall n \in [15]$.

We assume that in the subsequent cache assignment, users $\mathcal{U}_1 = (1, 2, 3, 4, 5, 6, 7, 8)$ are assigned to helper node 1, users $\mathcal{U}_2 = (9, 10, 11, 12, 13)$ to helper node 2 and users $\mathcal{U}_3 = (14, 15)$ to helper node 3. This corresponds to a cache occupancy vector $\mathbf{L} = (8, 5, 2)$. We also assume without loss of generality that the demand vector is $\mathbf{d} = (1, 2, \ldots, 15)$.

Delivery takes place in $|\mathcal{U}_1| = 8$ rounds, and each round will serve either $N_0 = 2$ users or no users from each of the following three ordered user groups

$$\begin{aligned}
\boldsymbol{s_1} &= \mathcal{U}_1 || \mathcal{U}_1 = (1, 2, \ldots, 7, 8, 1, 2, \ldots, 7, 8), \\
\boldsymbol{s_2} &= \mathcal{U}_2 || \mathcal{U}_2 = (9, 10, 11, 12, 13, 9, 10, 11, 12, 13), \\
\boldsymbol{s_3} &= \mathcal{U}_3 || \mathcal{U}_3 = (14, 15, 14, 15).
\end{aligned}$$

Specifically, rounds 1 through 8, will respectively serve the following sets of users

$$\begin{aligned}
\mathcal{R}_1 &= \{1, 2, 9, 10, 14, 15\} \\
\mathcal{R}_2 &= \{3, 4, 11, 12, 14, 15\} \\
\mathcal{R}_3 &= \{5, 6, 13, 9\} \\
\mathcal{R}_4 &= \{7, 8, 10, 11\} \\
\mathcal{R}_5 &= \{1, 2, 12, 13\} \\
\mathcal{R}_6 &= \{3, 4\} \\
\mathcal{R}_7 &= \{5, 6\} \\
\mathcal{R}_8 &= \{7, 8\}.
\end{aligned}$$

[8] This separable approach though can indeed work (cf. [7]) if each user has its own cache.

Before transmission, each requested subfile $W_{\mathcal{T}}^{(n)}$ is further split into $N_0 = 2$ mini-files $W_{\mathcal{T},1}^{(n)}$ and $W_{\mathcal{T},2}^{(n)}$. As noted in the general description of the scheme, the transmitted vector structure within each round, draws from [33] as it employs the linear combination of ZF-precoded vectors. In the first round, the server transmits, one after the other, the following 3 vectors

$$\boldsymbol{x}_{\{1,2,9,10\}} = \mathbf{H}_{\{1,2\}}^{-1} \begin{bmatrix} W_{2,1}^{(1)} \\ W_{2,1}^{(2)} \end{bmatrix} + \mathbf{H}_{\{9,10\}}^{-1} \begin{bmatrix} W_{1,1}^{(9)} \\ W_{1,1}^{(10)} \end{bmatrix} \tag{3.20}$$

$$\boldsymbol{x}_{\{1,2,14,15\}} = \mathbf{H}_{\{1,2\}}^{-1} \begin{bmatrix} W_{3,1}^{(1)} \\ W_{3,1}^{(2)} \end{bmatrix} + \mathbf{H}_{\{14,15\}}^{-1} \begin{bmatrix} W_{1,1}^{(14)} \\ W_{1,1}^{(15)} \end{bmatrix} \tag{3.21}$$

$$\boldsymbol{x}_{\{9,10,14,15\}} = \mathbf{H}_{\{9,10\}}^{-1} \begin{bmatrix} W_{3,1}^{(9)} \\ W_{3,1}^{(10)} \end{bmatrix} + \mathbf{H}_{\{14,15\}}^{-1} \begin{bmatrix} W_{2,1}^{(14)} \\ W_{2,1}^{(15)} \end{bmatrix} \tag{3.22}$$

where $\mathbf{H}_{\{i,j\}}^{-1}$ is the zero-forcing (ZF) precoder that inverts the channel $\mathbf{H}_{\{i,j\}} = [\mathbf{h}_i^T \mathbf{h}_j^T]$ from the transmitter to users i and j. Instead of ZF, one can naturally use a similar precoder with potentially better performance in different SNR ranges; we will consider the finite SNR regime in Chapter 6. To see how decoding takes place, let us first focus on users 1 and 2 during the transmission of $\boldsymbol{x}_{\{1,2,9,10\}}$, where we see that, due to ZF precoding, the users' respective received signals take the form

$$y_1 = W_{2,1}^{(1)} + \underbrace{\mathbf{h}_1^T \mathbf{H}_{\{9,10\}}^{-1} \begin{bmatrix} W_{1,1}^{(9)} \\ W_{1,1}^{(10)} \end{bmatrix}}_{\text{interference}} + w_1 \tag{3.23}$$

$$y_2 = W_{2,1}^{(2)} + \underbrace{\mathbf{h}_2^T \mathbf{H}_{\{9,10\}}^{-1} \begin{bmatrix} W_{1,1}^{(9)} \\ W_{1,1}^{(10)} \end{bmatrix}}_{\text{interference}} + w_2. \tag{3.24}$$

Users 1 and 2 use their cached content in cache node 1, to remove files $W_{1,1}^{(9)}, W_{1,1}^{(10)}$, and can thus directly decode their own desired subfiles. The same procedure is applied to the remaining users served in the first round.

Similarly, in the second round, we have

$$\boldsymbol{x}_{\{3,4,11,12\}} = \mathbf{H}_{\{3,4\}}^{-1} \begin{bmatrix} W_{2,1}^{(3)} \\ W_{2,1}^{(4)} \end{bmatrix} + \mathbf{H}_{\{11,12\}}^{-1} \begin{bmatrix} W_{1,1}^{(11)} \\ W_{1,1}^{(12)} \end{bmatrix} \tag{3.25}$$

$$\boldsymbol{x}_{\{3,4,14,15\}} = \mathbf{H}_{\{3,4\}}^{-1} \begin{bmatrix} W_{3,1}^{(3)} \\ W_{3,1}^{(4)} \end{bmatrix} + \mathbf{H}_{\{14,15\}}^{-1} \begin{bmatrix} W_{1,2}^{(14)} \\ W_{1,2}^{(15)} \end{bmatrix} \tag{3.26}$$

$$\boldsymbol{x}_{\{11,12,14,15\}} = \mathbf{H}_{\{11,12\}}^{-1} \begin{bmatrix} W_{3,1}^{(11)} \\ W_{3,1}^{(12)} \end{bmatrix} + \mathbf{H}_{\{14,15\}}^{-1} \begin{bmatrix} W_{2,2}^{(14)} \\ W_{2,2}^{(15)} \end{bmatrix} \tag{3.27}$$

and again in each round, each pair of users can cache-out some of the files, and then decode their own file due to the ZF precoder.

The next three transmissions, corresponding to the third round, are as follows

$$\boldsymbol{x}_{\{5,6,13,9\}}=\mathbf{H}^{-1}_{\{5,6\}}\begin{bmatrix}W^{(5)}_{2,1}\\W^{(6)}_{2,1}\end{bmatrix}+\mathbf{H}^{-1}_{\{13,9\}}\begin{bmatrix}W^{(13)}_{1,1}\\W^{(9)}_{1,2}\end{bmatrix} \tag{3.28}$$

$$\boldsymbol{x}_{\{5,6\}}=\mathbf{H}^{-1}_{\{5,6\}}\begin{bmatrix}W^{(5)}_{3,1}\\W^{(6)}_{3,1}\end{bmatrix} \tag{3.29}$$

$$\boldsymbol{x}_{\{13,9\}}=\mathbf{H}^{-1}_{\{13,9\}}\begin{bmatrix}W^{(13)}_{3,1}\\W^{(9)}_{3,2}\end{bmatrix} \tag{3.30}$$

where the transmitted vectors $\boldsymbol{x}_{\{5,6\}}$ and $\boldsymbol{x}_{\{13,9\}}$ simply use zero-forcing. Similarly round 4 serves the users in $\mathcal{R}_4$ by sequentially sending

$$\boldsymbol{x}_{\{7,8,10,11\}}=\mathbf{H}^{-1}_{\{7,8\}}\begin{bmatrix}W^{(7)}_{2,1}\\W^{(8)}_{2,1}\end{bmatrix}+\mathbf{H}^{-1}_{\{10,11\}}\begin{bmatrix}W^{(10)}_{1,2}\\W^{(11)}_{1,2}\end{bmatrix} \tag{3.31}$$

$$\boldsymbol{x}_{\{7,8\}}=\mathbf{H}^{-1}_{\{7,8\}}\begin{bmatrix}W^{(7)}_{3,1}\\W^{(8)}_{3,1}\end{bmatrix} \tag{3.32}$$

$$\boldsymbol{x}_{\{10,11\}}=\mathbf{H}^{-1}_{\{10,11\}}\begin{bmatrix}W^{(10)}_{3,2}\\W^{(11)}_{3,2}\end{bmatrix} \tag{3.33}$$

and round 5 serves the users in $\mathcal{R}_5$ by sequentially sending

$$\boldsymbol{x}_{\{1,2,12,13\}}=\mathbf{H}^{-1}_{\{1,2\}}\begin{bmatrix}W^{(1)}_{2,2}\\W^{(2)}_{2,2}\end{bmatrix}+\mathbf{H}^{-1}_{\{12,13\}}\begin{bmatrix}W^{(12)}_{1,2}\\W^{(13)}_{1,2}\end{bmatrix} \tag{3.34}$$

$$\boldsymbol{x}_{\{1,2\}}=\mathbf{H}^{-1}_{\{1,2\}}\begin{bmatrix}W^{(1)}_{3,2}\\W^{(2)}_{3,2}\end{bmatrix} \tag{3.35}$$

$$\boldsymbol{x}_{\{12,13\}}=\mathbf{H}^{-1}_{\{12,13\}}\begin{bmatrix}W^{(12)}_{3,2}\\W^{(13)}_{3,2}\end{bmatrix}. \tag{3.36}$$

Finally, for the remaining rounds $6,7,8$ which respectively involve user sets $\mathcal{R}_6,\mathcal{R}_7$ and $\mathcal{R}_8$ that are connected to the same helper cache 1, data is delivered using the following standard ZF-precoded transmissions

$$\boldsymbol{x}_{\{3,4\}}=\mathbf{H}^{-1}_{\{3,4\}}\begin{bmatrix}W^{(3)}_{2,2}||W^{(3)}_{3,2}\\W^{(4)}_{2,2}||W^{(4)}_{3,2}\end{bmatrix} \tag{3.37}$$

$$\boldsymbol{x}_{\{5,6\}}=\mathbf{H}^{-1}_{\{5,6\}}\begin{bmatrix}W^{(5)}_{2,2}||W^{(5)}_{3,2}\\W^{(6)}_{2,2}||W^{(6)}_{3,2}\end{bmatrix} \tag{3.38}$$

$$\boldsymbol{x}_{\{7,8\}}=\mathbf{H}^{-1}_{\{7,8\}}\begin{bmatrix}W^{(7)}_{2,2}||W^{(7)}_{3,2}\\W^{(8)}_{2,2}||W^{(8)}_{3,2}\end{bmatrix}. \tag{3.39}$$

The overall delivery time required to serve all users is

$$T = \frac{1}{6} \cdot 15 + \frac{1}{3} \cdot 3 = \frac{21}{6}$$

where the first summand is for rounds 1 through 5, and the second summand is for rounds 6 through 8. It is easy to see that this delay remains the same — given again worst-case demand vectors — for any user-to-cache association $\mathcal{U}$ with the same cache occupancy vector $\mathbf{L} = (8, 5, 2)$. Every time, this delay matches the converse

$$T^*((8,5,2)) \geq \frac{\sum_{r=1}^{2} L_r \binom{3-r}{1}}{2\binom{3}{1}} = \frac{8 \cdot 2 + 5 \cdot 1}{6} = \frac{21}{6} \tag{3.40}$$

of Theorem 2.

3.4 Information Theoretic Converse

Toward proving Theorems 1 and 2, we develop a lower bound on the normalized delivery time in (3.2) for each given cache occupancy vector $\mathbf{L}$. The proof technique is based on the breakthrough in [52] which — for the case of $\Lambda = K$, where each user has their own cache — employed index coding to bound the performance of coded caching. Part of the challenge here will be to account for having shared caches, and mainly to adapt the index coding approach to reflect non-uniform cache occupancy vectors. We recall that, in order to facilitate the reader to understand the main idea behind the general converse, an example of the construction of the converse is presented in Appendix A.1. We will begin with lower bounding the normalized delivery time $T^*(\mathcal{U}, \mathbf{d}, \mathbf{Z})$, for any user-to-cache association $\mathcal{U}$, demand vector $\mathbf{d}$ and a generic uncoded caching strategy $\mathbf{Z}$.

Identifying the distinct problems The caching problem is defined when the user-to-cache association $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda=1}^{\Lambda}$ and demand vector $\mathbf{d}$ are revealed. What we can easily see is that there are many combinations of $\{\mathcal{U}_\lambda\}_{\lambda=1}^{\Lambda}$ and $\mathbf{d}$ that jointly result in the same coded caching problem. After all, any permutation of the file indices requested by users assigned to the same cache, will effectively result in the same coded caching problem. As one can see, every *distinct* coded caching problem is fully defined by $\{\mathbf{d_\lambda}\}_{\lambda=1}^{\Lambda}$, where $\mathbf{d_\lambda}$ denotes the vector of file indices requested by the users in $\mathcal{U}_\lambda$, i.e., requested by the $|\mathcal{U}_\lambda|$ users associated to cache λ. The analysis is facilitated by reordering the demand vector $\mathbf{d}$ to take the form

$$\mathbf{d}(\mathcal{U}) \triangleq (\mathbf{d_1}, \ldots, \mathbf{d_\Lambda}). \tag{3.41}$$

Based on this, we define the set of worst-case demands associated to a given cache occupancy vector $\mathbf{L}$, to be

$$\mathcal{D}_\mathbf{L} = \{\mathbf{d}(\mathcal{U}) : \mathbf{d} \in \mathcal{D}_{wc}, \mathcal{U} \in \mathcal{U}_\mathbf{L}\}$$

where $\mathcal{D}_{wc}$ is the set of demand vectors $\mathbf{d}$ whose K entries are all different (i.e., where $d_i \neq d_j,\ i, j \in [\Lambda],\ i \neq j$, corresponding to the case where all users request different files). We will convert each such coded caching problem into an index coding problem.

The corresponding index coding problem To make the transition to the index coding problem, each requested file $W^{(\mathbf{d}_{\boldsymbol{\lambda}}(j))}$ is split into 2^{Λ} disjoint subfiles $W_{\mathcal{T}}^{(\mathbf{d}_{\boldsymbol{\lambda}}(j))}, \mathcal{T} \in 2^{[\Lambda]}$ where $\mathcal{T} \subset [\Lambda]$ indicates the set of helper nodes in which $W_{\mathcal{T}}^{(\mathbf{d}_{\boldsymbol{\lambda}}(j))}$ is cached[9]. Then — in the context of index coding — each subfile $W_{\mathcal{T}}^{(\mathbf{d}_{\boldsymbol{\lambda}}(j))}$ can be seen as being requested by a different user that has as side information all the content $\mathcal{Z}_\lambda$ of the same helper node λ. Naturally, no subfile of the form $W_{\mathcal{T}}^{(\mathbf{d}_{\boldsymbol{\lambda}}(j))}$, $\mathcal{T} \ni \lambda$ is requested, because helper node λ already has this subfile. Therefore the corresponding index coding problem is defined by $K2^{\Lambda-1}$ requested subfiles, and it is fully represented by the side-information graph $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$, where $\mathcal{V}_\mathcal{G}$ is the set of vertices (each vertex/node representing a different subfile $W_{\mathcal{T}}^{(\mathbf{d}_{\boldsymbol{\lambda}}(j))}, \mathcal{T} \not\ni \lambda$) and $\mathcal{E}_\mathcal{G}$ is the set of direct edges of the graph. Following standard practice in index coding, a directed edge from node $W_{\mathcal{T}}^{(\mathbf{d}_{\boldsymbol{\lambda}}(j))}$ to $W_{\mathcal{T}'}^{(\mathbf{d}_{\lambda'}(j'))}$ exists if and only if $\lambda' \in \mathcal{T}$. For any given $\mathcal{U}$, $\mathbf{d}$ (and of course, for any caching scheme $\mathbf{Z}$) the total delay T required for this index coding problem, is the completion time for the corresponding coded caching problem.

Lower bounding $T^*(\mathcal{U}, \mathbf{d}, \mathbf{Z})$ We are interested in lower bounding $T^*(\mathcal{U}, \mathbf{d}, \mathbf{Z})$ which represents the total delay required to serve the users for the index coding problem corresponding to the side-information graph $\mathcal{G}_{\mathcal{U},\mathbf{d}}$ defined by $\mathcal{U}, \mathbf{d}, \mathbf{Z}$ or equivalently by $\mathbf{d}(\mathcal{U}), \mathbf{Z}$.

In the next lemma, we remind the reader — in the context of our setting — the index-coding converse from [54].

Lemma 1. *(Cut-set-type converse [54]) For a given $\mathcal{U}, \mathbf{d}, \mathbf{Z}$, in the corresponding side information graph $\mathcal{G}_{\mathcal{U},\mathbf{d}} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ of the N_0-antenna MISO broadcast channel with $\mathcal{V}_\mathcal{G}$ vertices/nodes and $\mathcal{E}_\mathcal{G}$ edges, the following inequality holds*

$$T \geq \frac{1}{N_0} \sum_{\mathcal{V} \in \mathcal{V}_\mathcal{J}} |\mathcal{V}| \tag{3.42}$$

for every acyclic induced subgraph $\mathcal{J}$ of $\mathcal{G}_{\mathcal{U},\mathbf{d}}$, where $\mathcal{V}_\mathcal{J}$ denotes the set of nodes of the subgraph $\mathcal{J}$, and where $|\mathcal{V}|$ is the size of the message/subfile/node $\mathcal{V}$.

Proof. The above lemma draws from [54, Corollary 1] (see also [55, Corollary 2] for a simplified version), and is proved in the Appendix Section A.2.1.

Creating large acyclic subgraphs Lemma 1 suggests the need to create (preferably large) acyclic subgraphs of $\mathcal{G}_{\mathcal{U},\mathbf{d}}$. The following lemma describes how to properly choose a set of nodes to form a large acyclic subgraph.

[9]Notice that by considering a subpacketization based on the power set $2^{[\Lambda]}$, and by allowing for any possible size of these subfiles, the generality of the result is preserved. Naturally, this does not impose any subpacketization related performance issues because this is done only for the purpose of creating a converse.

Lemma 2. *Consider a subgraph $\mathcal{J}$ of $\mathcal{G}_{\mathcal{U},\mathbf{d}}$ corresponding to the index coding problem defined by $\mathcal{U}, \mathbf{d}, \mathbf{Z}$ for any $\mathcal{U}$ with cache occupancy vector $\mathbf{L}$, consisting of all subfiles $W_{\mathcal{T}_\lambda}^{(\mathbf{d}_{\sigma_s(\lambda)}(j))}$, $\forall j \in [L_\lambda]$, $\forall \lambda \in [\Lambda]$ for all $\mathcal{T}_\lambda \subseteq [\Lambda] \setminus \{\sigma_s(1), \dots, \sigma_s(\lambda)\}$ where $\sigma_s \in S_\Lambda$ is the permutation such that $|\mathcal{U}_{\sigma_s(1)}| \geq |\mathcal{U}_{\sigma_s(2)}| \geq \cdots \geq |\mathcal{U}_{\sigma_s(\Lambda)}|$. This subgraph is acyclic.*

Proof. The proof, which can be found in the Appendix Section A.2.2, is an adaptation of [52, Lemma 1] to the current setting.

Remark 3. *The choice of the permutation σ_s is critical for the development of a tight converse. Any other choice $\sigma \in S_\Lambda$ may result — in some crucial cases — in an acyclic subgraph with a smaller number of nodes and therefore a looser bound. This approach here deviates from the original approach in [52, Lemma 1], which instead considered — for each $\mathbf{d}, \mathbf{Z}$, for the uniform user-to-cache association case of $K = \Lambda$ — the set of* all *possible permutations, that jointly resulted in a certain symmetry that is crucial to that proof. Here in our case, such symmetry would not serve the same purpose as it would dilute the non-uniformity in $\mathbf{L}$ that we are trying to capture. Our choice of a single carefully chosen permutation, allows for a bound which — as it turns out — is tight even in non-uniform cases. The reader is also referred to Section A.1 for an explanatory example.*

Having chosen an acyclic subgraph according to Lemma 2, we return to Lemma 1 and form — by adding the sizes of all subfiles associated to the chosen acyclic graph — the following lower bound

$$T^*(\mathcal{U}, \mathbf{d}, \mathbf{Z}) \geq T^{LB}(\mathcal{U}, \mathbf{d}, \mathbf{Z}) \tag{3.43}$$

where

$$\begin{aligned} T^{LB}(\mathcal{U}, \mathbf{d}, \mathbf{Z}) \triangleq \frac{1}{N_0} \Bigg(& \sum_{j=1}^{L_{\sigma_s(1)}} \sum_{\mathcal{T}_1 \subseteq [\Lambda] \setminus \{\sigma_s(1)\}} |W_{\mathcal{T}_1}^{(\mathbf{d}_{\sigma_s(1)}(j))}| \\ & + \sum_{j=1}^{L_{\sigma_s(2)}} \sum_{\mathcal{T}_2 \subseteq [\Lambda] \setminus \{\sigma_s(1), \sigma_s(2)\}} |W_{\mathcal{T}_2}^{(\mathbf{d}_{\sigma_s(2)}(j))}| + \dots \\ & + \sum_{j=1}^{L_{\sigma_s(\Lambda)}} \sum_{\mathcal{T}_\Lambda \subseteq [\Lambda] \setminus \{\sigma_s(1), \dots, \sigma_s(\Lambda)\}} |W_{\mathcal{T}_\Lambda}^{\mathbf{d}_{(\sigma_s(\Lambda)}(j))}| \Bigg). \end{aligned} \tag{3.44}$$

Our interest lies in a lower bound for the worst-case delivery time/delay associated to cache occupancy vector $\mathbf{L}$. Such a worst-case naturally corresponds to the scenario where all users request different files, i.e., where all the entries of the demand vector $\mathbf{d}(\mathcal{U})$ are different. The corresponding lower bound can be developed by averaging over worst-case demands. Recalling our set $\mathcal{D}_{\mathbf{L}}$, the worst-case delivery time can thus be written as

$$T^*(\mathbf{L}) \triangleq \min_{\mathbf{Z}} \overbrace{\max_{(\mathcal{U}, \mathbf{d}) \in (\mathcal{U}_{\mathbf{L}}, [N]^K)} T(\mathcal{U}, \mathbf{d}, \mathbf{Z})}^{T(\mathbf{L}, \mathbf{Z})} \tag{3.45}$$

$$\overset{(a)}{\geq} \min_{\mathbf{Z}} \frac{1}{|\mathcal{D}_{\mathbf{L}}|} \sum_{\mathbf{d}(\mathcal{U})\in\mathcal{D}_{\mathbf{L}}} T(\mathbf{d}(\mathcal{U}),\mathbf{Z}) \tag{3.46}$$

where in step (a), we used the following change of notation $T(\mathbf{d}(\mathcal{U}),\mathbf{Z}) \triangleq T(\mathcal{U},\mathbf{d},\mathbf{Z})$ and averaged over worst-case demands.

With a given cache occupancy vector $\mathbf{L}$ in mind, in order to construct $\mathcal{D}_{\mathbf{L}}$ (so that we can then average over it), we will consider all demand vectors $\mathbf{d} \in \mathcal{D}_{wc}$ for all permutations $\pi \in S_\Lambda$. Then for each $\mathbf{d}$, we create the following set of Λ vectors

$$\begin{aligned}
\mathbf{d_1'} &= (d_1 : d_{L_1}),\\
\mathbf{d_2'} &= (d_{L_1+1} : d_{L_1+L_2}),\\
&\vdots\\
\mathbf{d_\Lambda'} &= (d_{\sum_{i=1}^{\Lambda-1} L_i\ +1} : d_K)
\end{aligned}$$

and for each permutation $\pi \in S_\Lambda$ applied to the set $\{1,2,\ldots,\Lambda\}$, a demand vector $\mathbf{d}(\mathcal{U})$ is constructed as follows

$$\mathbf{d}(\mathcal{U}) \triangleq (\mathbf{d_1},\mathbf{d_2},\ldots,\mathbf{d_\Lambda}) \tag{3.47}$$

$$= (\mathbf{d'_{\pi^{-1}(1)}},\mathbf{d'_{\pi^{-1}(2)}},\ldots,\mathbf{d'_{\pi^{-1}(\Lambda)}}), \tag{3.48}$$

where $\pi^{-1}(\cdot)$ is the inverse function of the permutation operation.

This procedure is repeated for all $\Lambda!$ permutations $\pi \in S_\Lambda$ and all $P(N,K)$ worst-case demands $\mathbf{d} \in \mathcal{D}_{wc}$. This implies that the cardinality of $\mathcal{D}_{\mathbf{L}}$ is $|\mathcal{D}_{\mathbf{L}}| = P(N,K)\cdot\Lambda!$.

Using this designed set $\mathcal{D}_{\mathbf{L}}$, now the optimal worst-case delivery time in (3.46) is bounded as

$$T^*(\mathbf{L}) = \min_{\mathbf{Z}} T^*(\mathbf{L},\mathbf{Z}) \tag{3.49}$$

$$\geq \min_{\mathbf{Z}} \frac{1}{P(N,K)\Lambda!} \sum_{\mathbf{d}(\mathcal{U})\in\mathcal{D}_{\mathbf{L}}} T^{LB}(\mathbf{d}(\mathcal{U}),\mathbf{Z}) \tag{3.50}$$

where $T^{LB}(\mathbf{d}(\mathcal{U}),\mathbf{Z})$ is given by (3.44) for each reordered demand vector $\mathbf{d}(\mathcal{U}) \in \mathcal{D}_{\mathbf{L}}$. Rewriting the summation in (3.50), we get

$$\sum_{\mathbf{d}(\mathcal{U})\in\mathcal{D}_{\mathbf{L}}} T^{LB}(\mathbf{d}(\mathcal{U}),\mathbf{Z}) =$$

$$= \frac{1}{N_0}\sum_{i=0}^{\Lambda}\sum_{n\in[N]}\sum_{\mathcal{T}\subseteq[\Lambda]:|\mathcal{T}|=i} |W_{\mathcal{T}}^{(n)}| \cdot \underbrace{\sum_{\mathbf{d}(\mathcal{U})\in\mathcal{D}_{\mathbf{L}}} \mathbb{1}_{\mathcal{V}_{\mathcal{J}_s^{\mathbf{d}(\mathcal{U})}}}(W_{\mathcal{T}}^{(n)})}_{\triangleq Q_i(W_{\mathcal{T}}^{(n)})}, \tag{3.51}$$

where $\mathcal{V}_{\mathcal{J}_s^{\mathbf{d}(\mathcal{U})}}$ is the set of vertices in the acyclic subgraph chosen according to Lemma 2 for a given $\mathbf{d}(\mathcal{U})$. In the above, $\mathbb{1}_{\mathcal{V}_{\mathcal{J}_s^{\mathbf{d}(\mathcal{U})}}}(W_{\mathcal{T}}^{(n)})$ denotes the indicator function which takes the value of 1 only if $W_{\mathcal{T}}^{(n)} \subset \mathcal{V}_{\mathcal{J}_s^{\mathbf{d}(\mathcal{U})}}$, else it is set to zero.

A crucial step toward removing the dependence on $\mathcal{T}$, comes from the fact that

$$\begin{aligned} Q_i = Q_i(W_{\mathcal{T}}^{(n)}) &\triangleq \sum_{\mathbf{d}(\mathcal{U})\in\mathcal{D}_{\mathbf{L}}} \mathbb{1}_{\mathcal{V}_{\mathcal{J}_s^{\mathbf{d}(\mathcal{U})}}}(W_{\mathcal{T}}^{(n)}) \\ &= \binom{N-1}{K-1} \sum_{r=1}^{\Lambda} P(\Lambda-i-1, r-1)(\Lambda-r)! L_r \\ &\times P(K-1, L_r-1)(K-L_r)!(\Lambda-i) \end{aligned} \tag{3.52}$$

where we can see that the total number of times a specific subfile appears — in the summation in (3.51), over the set of all possible $\mathbf{d}(\mathcal{U}) \in \mathcal{D}_{\mathbf{L}}$, and given our chosen permutation σ_s — is not dependent on the subfile itself but is dependent only on the number of caches $i = |\mathcal{T}|$ storing that subfile. The proof of (3.52) can be found in Section A.2.3.

In the spirit of [52], defining

$$x_i \triangleq \sum_{n\in[N]} \sum_{\mathcal{T}\subseteq[\Lambda]:|\mathcal{T}|=i} |W_{\mathcal{T}}^{(n)}| \tag{3.53}$$

to be the total amount of data stored in exactly i helper nodes, we see that

$$N = \sum_{i=0}^{\Lambda} x_i = \sum_{i=0}^{\Lambda} \sum_{n\in[N]} \sum_{\mathcal{T}\subseteq[\Lambda]:|\mathcal{T}|=i} |W_{\mathcal{T}}^{(n)}| \tag{3.54}$$

and we see that combining (3.50), (3.51) and (3.52), gives

$$T^*(\mathbf{L}, \mathbf{Z}) \geq \frac{1}{N_0} \sum_{i=0}^{\Lambda} \frac{Q_i}{P(N,K)\Lambda!} x_i. \tag{3.55}$$

Now substituting (3.52) into (3.55), after some algebraic manipulations, we get that

$$T^*(\mathbf{L}, \mathbf{Z}) \geq \frac{1}{N_0} \sum_{i=0}^{\Lambda} \frac{\sum_{r=1}^{\Lambda-i} L_r \binom{\Lambda-r}{i}}{N\binom{\Lambda}{i}} x_i \tag{3.56}$$

$$= \frac{1}{N_0} \sum_{i=0}^{\Lambda} \frac{x_i}{N} c_i \tag{3.57}$$

where $c_i \triangleq \frac{\sum_{r=1}^{\Lambda-i} L_r \binom{\Lambda-r}{i}}{\binom{\Lambda}{i}}$ decreases with $i \in [\Lambda]_0$. The proof of the transition from (3.55) to (3.56), as well as the monotonicity proof for the sequence $\{c_i\}_{i\in[\Lambda]_0}$, are given in Appendix Sections A.2.4 and A.2.5 respectively.

Under the file-size constraint given in (3.54), and given the following cache-size constraint

$$\sum_{i=0}^{\Lambda} i \cdot x_i \leq \Lambda M \tag{3.58}$$

the expression in (3.56) serves as a lower bound on the delay of any caching scheme $\mathbf{Z}$ which implies a set of $\{x_i\}$.

We then employ the Jensen's-inequality based technique of [5, Proof of Lemma 2] to minimize the expression in (3.56), over all admissible $\{x_i\}$. Hence for any integer $\Lambda\gamma$, we have

$$T^*(\mathbf{L},\mathbf{Z}) \geq \frac{\sum_{r=1}^{\Lambda-\Lambda\gamma} L_r \binom{\Lambda-r}{\Lambda\gamma}}{\binom{\Lambda}{\Lambda\gamma}} \tag{3.59}$$

whereas for all other values of $\Lambda\gamma$, this is extended to its convex lower envelop. The detailed derivation of (3.59) can again be found in Appendix Section A.2.6.

This concludes lower bounding $\max_{(\mathcal{U},\mathbf{d})\in(\mathcal{U}_{\mathbf{L}},[N]^K)} T(\mathcal{U},\mathbf{d},\mathbf{Z})$, and thus — given that the right hand side of (3.59) is independent of $\mathbf{Z}$ — lower bounds the performance for any scheme $\mathbf{Z}$, which hence concludes the proof of the converse for Theorem 2 (and consequently for Theorem 1 after setting $N_0 = 1$). □

Proof of the Converse for (3.5a)

For the uniform case of $\mathbf{L} = \left[\frac{K}{\Lambda}, \frac{K}{\Lambda}, \ldots, \frac{K}{\Lambda}\right]$, the lower bound in (3.59) becomes

$$\frac{1}{N_0}\frac{\sum_{r=1}^{\Lambda-\Lambda\gamma} L_r \binom{\Lambda-r}{\Lambda\gamma}}{\binom{\Lambda}{\Lambda\gamma}} = \frac{1}{N_0}\frac{K}{\Lambda}\frac{\sum_{r=1}^{\Lambda-\Lambda\gamma} \binom{\Lambda-r}{\Lambda\gamma}}{\binom{\Lambda}{\Lambda\gamma}} \tag{3.60}$$

$$\overset{(a)}{=} \frac{1}{N_0}\frac{K}{\Lambda}\frac{\binom{\Lambda}{\Lambda\gamma+1}}{\binom{\Lambda}{\Lambda\gamma}} \tag{3.61}$$

$$= \frac{K}{N_0}\frac{(1-\gamma)}{(\Lambda\gamma+1)} \tag{3.62}$$

where the equality in step (a) is due to Pascal's triangle which says that $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$. □

3.5 Achievable Scheme for the Uniform Setting with $N_0 \geq \frac{K}{\Lambda}$

In this section we will describe the algorithm that achieves the delivery time in equation (3.5b). We begin with describing the algorithm through the use of an example and continue with the general algorithm, which requires that N_0 is an integer multiple of $\frac{K}{\Lambda}$.

3.5.1 Illustrative Example

Consider a scenario where a base station equipped with $N_0 = 4$ antennas has access to a library with $N = 8$ files $W^{(1)}, W^{(2)}, \ldots, W^{(8)}$, and is connected to $K = 8$ users. We assume that the number of caches is $\Lambda = 4$ and that users are distributed uniformly among the caches, such that each cache serves $\frac{K}{\Lambda} = 2$ users. Furthermore, we also assume that the per-cache capacity is $M = 2$ (i.e., $\gamma = \frac{1}{4}$).

In the cache placement phase, each file $W^{(n)}, n \in [8]$ is split into $\binom{\Lambda}{\Lambda\gamma} = 4$ equally-sized subfiles denoted by $W_1^{(n)}, W_2^{(n)}, W_3^{(n)}, W_4^{(n)}$. Employing the cache placement algorithm of [2], we have that each cache $\lambda \in [4]$ stores $W_\lambda^{(n)}, \forall n \in [8]$. In the delivery phase, we further split each subfile $W_\tau^{(n)}$, $\tau \in [4]$ into 2 equally-sized and disjoint minifiles $W_{\tau,1}^{(n)}, W_{\tau,2}^{(n)}$. For simplicity, we will also use the notation $A \equiv W^{(1)}$, $B \equiv W^{(2)}$, $C \equiv W^{(3)}$, and so on.

Without loss of generality, we assume that users $\{1, 5\}$ are connected to the first cache and request files $\{A, E\}$, users $\{2, 6\}$ are connected to the second cache and request $\{B, F\}$, and so on.

For simplicity, in this example we will use the *one-shot* (see Section 3.5.4) variation of the proposed scheme in Section 3.5.3. The delivery algorithm consists of 4 rounds, each serving users from $\Lambda\gamma + \frac{N_0\Lambda}{K} = 3$ different caches.

In the first round, the server transmits to users in caches $1, 2, 3$ the vector

$$\mathbf{x}_{\{1,2,3,5,6,7\}} = \mathbf{H}_{1,5,3,7}^{-1} \begin{bmatrix} A_{2,1} \\ E_{2,1} \\ C_{2,1} \\ G_{2,1} \end{bmatrix} + \mathbf{H}_{1,5,2,6}^{-1} \begin{bmatrix} A_{3,1} \\ E_{3,1} \\ B_{3,1} \\ F_{3,1} \end{bmatrix} + \mathbf{H}_{2,6,3,7}^{-1} \begin{bmatrix} B_{1,1} \\ F_{1,1} \\ C_{1,1} \\ G_{1,1} \end{bmatrix} \tag{3.63}$$

in a time slot of normalized duration $\frac{2}{8}$. $\mathbf{H}_{i,j,p,q}^{-1}$ is the zero-forcing (ZF) precoder that inverts the channel matrix $\mathbf{H}_{i,j,p,q} \triangleq [\mathbf{h}_i^T \mathbf{h}_j^T \mathbf{h}_p^T \mathbf{h}_q^T]$. To describe the decoding, we focus on user 1 who receives the signal

$$y_1 = A_{2,1} + A_{3,1} + \underbrace{\mathbf{h}_1^T \cdot \mathbf{H}_{2,6,3,7}^{-1} \begin{bmatrix} B_{1,1} \\ F_{1,1} \\ C_{1,1} \\ G_{1,1} \end{bmatrix}}_{\text{interference}} + w_1. \tag{3.64}$$

We observe that user 1 is connected to cache 1 and that it has perfect knowledge of the channel state, thus user 1 can reconstruct the interference term in (3.64) and subtract it from y_1 to obtain (neglecting the noise) $\bar{y}_1 = A_{2,1} + A_{3,1}$. Recalling that $|A_{2,1}| = |A_{3,1}| = \frac{1}{8}$, which is half of the transmission duration for vector $\mathbf{v}_{1,2,3}$, user 1 can successfully decode the 2 desired minifiles $A_{2,1}, A_{3,1}$. The same decoding procedure is applied to the users served in the first round.

Similarly, in the other 3 rounds the transmitted vectors are

$$\mathbf{x}_{1,2,4,5,6,8} = \mathbf{H}_{1,5,4,8}^{-1} \begin{bmatrix} A_{2,2} \\ E_{2,2} \\ D_{2,1} \\ H_{2,1} \end{bmatrix} + \mathbf{H}_{1,5,2,6}^{-1} \begin{bmatrix} A_{4,1} \\ E_{4,1} \\ B_{4,1} \\ F_{4,1} \end{bmatrix} + \mathbf{H}_{2,6,4,8}^{-1} \begin{bmatrix} B_{1,2} \\ F_{1,2} \\ D_{1,1} \\ H_{1,1} \end{bmatrix}, \tag{3.65}$$

$$\mathbf{x}_{1,3,4,5,7,8} = \mathbf{H}_{1,5,4,8}^{-1} \begin{bmatrix} A_{3,2} \\ E_{3,2} \\ D_{3,1} \\ H_{3,1} \end{bmatrix} + \mathbf{H}_{1,5,3,7}^{-1} \begin{bmatrix} A_{4,2} \\ E_{4,2} \\ C_{4,1} \\ G_{4,1} \end{bmatrix} + \mathbf{H}_{4,8,3,7}^{-1} \begin{bmatrix} D_{1,2} \\ H_{1,2} \\ C_{1,2} \\ G_{1,2} \end{bmatrix}, \tag{3.66}$$

$$\mathbf{x}_{2,3,4,6,7,8} = \mathbf{H}_{2,6,4,8}^{-1} \begin{bmatrix} B_{3,2} \\ F_{3,2} \\ D_{3,2} \\ H_{3,2} \end{bmatrix} + \mathbf{H}_{2,6,3,7}^{-1} \begin{bmatrix} B_{4,2} \\ F_{4,2} \\ C_{4,2} \\ G_{4,2} \end{bmatrix} + \mathbf{H}_{3,7,4,8}^{-1} \begin{bmatrix} C_{2,2} \\ G_{2,2} \\ D_{2,2} \\ H_{2,2} \end{bmatrix}. \tag{3.67}$$

The overall optimal delivery time required to serve all users' demands is $T = \frac{2}{8} \cdot 4 = 1$, which corresponds to a sum degrees of freedom of $DoF = \frac{8(1-1/4)}{T} = 6 = K\gamma + N_0$.

We proceed with the description of the general scheme.

3.5.2 Cache Placement Scheme

The cache placement phase is the same as the one in [2], for a setting with Λ users, each with its own dedicated cache. Therefore, each file $W^{(n)}, n \in [N]$ is split into $\binom{\Lambda}{\Lambda\gamma}$ disjoint subfiles $W_\tau^{(n)}$, for each $\tau \subset [\Lambda], |\tau| = \Lambda\gamma$. Then, each cache λ stores a fraction γ of the library according to the following policy

$$\mathcal{Z}_\lambda = \{W_\tau^{(n)} : \tau \ni \lambda, \forall n \in [N]\}. \tag{3.68}$$

3.5.3 Delivery Scheme

Upon receiving the users' requests, the server further splits the demanded subfiles $W_\tau^{(d_k)}$ in $\binom{\Lambda-\Lambda\gamma-1}{\frac{N_0\Lambda}{K}-1}$ minifiles as follows

$$W_\tau^{(n)} = \left\{ W_{\tau,r}^{(n)} | r \in \left\{ 1, 2, \ldots, \binom{\Lambda - \Lambda\gamma - 1}{\frac{N_0\Lambda}{K} - 1} \right\} \right\}. \tag{3.69}$$

For each set of caches $\Phi \subseteq [\Lambda]$, each of cardinality $|\Phi| = \Lambda\gamma + \frac{N_0\Lambda}{K}$, the server transmits $\binom{\Lambda\gamma + \frac{N_0\Lambda}{K} - 1}{\Lambda\gamma}$ vectors of the form

$$\mathbf{x}_{\chi_\Phi}^{(i)} = \sum_{\phi \subset \Phi : |\phi| = \Lambda\gamma} c_{\Phi\backslash\phi}^{(i)} \cdot \mathbf{H}_{\Phi\backslash\phi}^{-1} \cdot \begin{bmatrix} \mathbf{W}_{\phi,r_1}^{\mathbf{d}_{\Phi\backslash\phi}(1)} \\ \mathbf{W}_{\phi,r_2}^{\mathbf{d}_{\Phi\backslash\phi}(2)} \\ \vdots \\ \mathbf{W}_{\phi,r_{\frac{N_0\Lambda}{K}}}^{\mathbf{d}_{\Phi\backslash\phi}\left(\frac{N_0\Lambda}{K}\right)} \end{bmatrix}, \quad i \in \left\{ 1, 2, \ldots, \binom{\Lambda\gamma + \frac{N_0\Lambda}{K} - 1}{\Lambda\gamma} \right\}, \tag{3.70}$$

In the above, χ_Φ is the set of users to which the message is transmitted, i.e. $\chi_\Phi \triangleq \cup_{\lambda \in \Phi} \mathcal{U}_\lambda$, $c_{\Phi\backslash\phi}^{(i)} \in \mathbb{C}$ denotes an arbitrary coefficient, $\Phi_{\backslash\phi}(l), l \in \left[\frac{N_0\Lambda}{K}\right]$ denotes the l-th element of the ordered set of caches $\Phi\backslash\phi$, where $\phi \subset \Phi : |\phi| = \Lambda\gamma$, and $\mathbf{W}_{\phi,r_l}^{\mathbf{d}_{\Phi\backslash\phi}(l)}$ denotes a $\frac{K}{\Lambda} \times 1$

vector of minifiles requested by all users connected to cache $\Phi_{\setminus\phi}(l)$, i.e.

$$\mathbf{W}_{\phi,r_l}^{\mathbf{d}_{\Phi_{\setminus\phi}(l)}} \triangleq \begin{bmatrix} W_{\phi,r_l}^{(\mathbf{d}_{\Phi_{\setminus\phi}(l)}(1))} \\ W_{\phi,r_l}^{(\mathbf{d}_{\Phi_{\setminus\phi}(l)}(2))} \\ \vdots \\ W_{\phi,r_l}^{(\mathbf{d}_{\Phi_{\setminus\phi}(l)}(\frac{K}{\Lambda}))} \end{bmatrix}.$$

The choice of indices $r_1, r_2, \ldots, r_{\frac{N_0\Lambda}{K}}$ is sequential, guaranteeing that no minifile is transmitted twice.

Decoding Directly from (3.70) and for a fixed Φ and i, the received signal at the $q-$th user, denoted by u, of cache $\lambda \in \Phi$ is

$$y_{u,\Phi}^{(i)} = \underbrace{\sum_{\phi\subset\Phi\setminus\{\lambda\}:|\phi|=\Lambda\gamma} c_{\Phi_{\setminus\phi}}^{(i)} W_{\phi,r}^{(d_u)}}_{\mathcal{L}_{\Phi,u}^{(i)}} + \iota_u^{(i)}$$

where $\mathcal{L}_{\Phi,u}^{(i)}$ is the part of the received signal useful for user u, while $\iota_u^{(i)}$ is an interference term that takes the form

$$\mathbf{h}_u^T \cdot \sum_{\phi\subset\Phi:\phi\ni\lambda,|\phi|=\Lambda\gamma} c_{\Phi_{\setminus\phi}}^{(i)} \cdot \mathbf{H}_{\Phi_{\setminus\phi}}^{-1} \cdot \begin{bmatrix} \mathbf{W}_{\phi,r_1}^{\mathbf{d}^{\Phi_{\setminus\phi}(1)}} \\ \mathbf{W}_{\phi,r_2}^{\mathbf{d}^{\Phi_{\setminus\phi}(2)}} \\ \vdots \\ \mathbf{W}_{\phi,r_{\frac{N_0\Lambda}{K}}}^{\mathbf{d}^{\Phi_{\setminus\phi}(\frac{N_0\Lambda}{K})}} \end{bmatrix} + w_u.$$

We can see that user u can reconstruct and remove (neglecting the noise) the interference term $\iota_u^{(i)}$, because it can obtain the minifiles contained in $\iota_u^{(i)}$ through its cache. After collecting the signals

$$\bar{y}_{u,\Phi}^{(i)} = y_{u,\Phi}^{(i)} - \iota_u^{(i)}, \quad \forall i \in \left[\binom{\Lambda\gamma + \frac{N_0\Lambda}{K} - 1}{\Lambda\gamma}\right],$$

user u will possess $\binom{\Lambda\gamma+\frac{N_0\Lambda}{K}-1}{\Lambda\gamma}$ linear combinations $\mathcal{L}_{\Phi,u}^{(i)}$ of the same set of desired minifiles $\Psi = \left\{W_{\phi,r_l}^{(u)} | \phi \in \Phi \setminus \{g\}, |\phi| = \Lambda\gamma\right\}$.

Directly from (3.70), we observe that $\mathbf{x}_{\chi_\Phi}^{(i)}$ is constructed as the sum of $\binom{\Lambda\gamma+\frac{N_0\Lambda}{K}}{\Lambda\gamma}$ vectors, each containing minifiles for users connected to the caches in the set $\Phi \setminus \phi$, where $\phi \subset \Phi : |\phi| = \Lambda\gamma$. Taking this into account, the number of times that a cache $\lambda \in \Phi$ appears in all the sets $\Phi \setminus \phi$ appearing in $\mathbf{v}_\Phi^{(i)}$ can be computed to be $\binom{\Lambda\gamma+\frac{N_0\Lambda}{K}-1}{\Lambda\gamma}$. This means that the vector $\mathbf{v}_\Phi^{(i)}$ contains $|\Psi| = \binom{\Lambda\gamma+\frac{N_0\Lambda}{K}-1}{\Lambda\gamma}$ minifiles for each user in cache λ.

As a result, user u can successfully decode all the desired files in Ψ from the $\binom{\Lambda\gamma+\frac{N_0\Lambda}{K}-1}{\Lambda\gamma}$ linear combinations $\mathcal{L}^{(i)}_{\Phi,u}$.

3.5.4 Calculation of the Normalized Delivery Time

We observe that the scheme splits each file into $S = \binom{\Lambda}{\Lambda\gamma}\binom{\Lambda-\Lambda\gamma-1}{\frac{n_0\Lambda}{K}-1}$ minifiles, where the first term follows from the subpacketization required by the cache placement in equation (3.5.2) and the second term is due to the further subpacketization needed by the delivery phase (cf. (3.69)).

Direcly from the scheme, the total number of transmissions can be easily computed as $\mathcal{N} = \binom{\Lambda}{\Lambda\gamma+\frac{N_0\Lambda}{K}}\binom{\Lambda\gamma+\frac{N_0\Lambda}{K}-1}{\Lambda\gamma}$. Finally, we notice that all transmissions have the same duration of $\frac{1}{S}$, yielding a total delivery time of

$$T = \frac{\binom{\Lambda}{\Lambda\gamma+\frac{N_0\Lambda}{K}}\binom{\Lambda\gamma+\frac{N_0\Lambda}{K}-1}{\Lambda\gamma}}{\binom{\Lambda}{\Lambda\gamma}\binom{\Lambda-\Lambda\gamma-1}{\frac{N_0\Lambda}{K}-1}} = \frac{K(1-\gamma)}{K\gamma+N_0}.$$

A one-shot linear variation of the delivery scheme

In this subsection, we present a variation of the delivery phase presented in Section 3.5.3. Unlike the previous algorithm, this scheme has the *one-shot* property, where each part of the requested messages is transmitted only once.

In the new scheme, each transmission associated to Φ occurs in a time slot of duration $T_s = \frac{\binom{\Lambda\gamma+\frac{N_0\Lambda}{K}-1}{\Lambda\gamma}}{S}$, where the server transmits the message

$$\mathbf{x}_{\chi\Phi} = \sum_{\phi\subset\Phi:|\phi|=\Lambda\gamma} \mathbf{H}^{-1}_{\Phi\setminus\phi} \cdot \begin{bmatrix} \mathbf{W}^{\mathbf{d}}_{\phi,r_1}{}^{\Phi\setminus\phi(1)} \\ \mathbf{W}^{\mathbf{d}}_{\phi,r_2}{}^{\Phi\setminus\phi(2)} \\ \vdots \\ \mathbf{W}^{\mathbf{d}}_{\phi,r_{\frac{L\Lambda}{K}}}{}^{\Phi\setminus\phi\left(\frac{N_0\Lambda}{K}\right)} \end{bmatrix} \tag{3.71}$$

of duration $|\mathbf{x}_{\chi\Phi}| = \frac{1}{S}$. After receiving the signal $\mathbf{h}_u^T\mathbf{x}_{\chi\Phi}$, user u removes the interference term as described in Section 3.5.3. This step presents user u with a multiple access channel (MAC) with $|\Psi| = \binom{\Lambda\gamma+\frac{N_0\Lambda}{K}-1}{\Lambda\gamma}$ messages to resolve. Having the slot duration T_s be $\binom{\Lambda\gamma+\frac{N_0\Lambda}{K}-1}{\Lambda\gamma}$ times larger than the message duration $|\mathbf{x}_{\chi\Phi}|$, guarantees that we are within the achievable rate region of the multiple access channel (MAC). The rest of the calculations follow as before.

3.6 Follow-Up Works

As a testament to the importance of this shared caches setting, it is worth noting that after the preliminary conference version of some of the results presented in this chapter appeared in [41], the shared-cache problem for the single-antenna setting studied in this chapter, has been extended in a variety of ways. For example, in [56] the authors extended the single antenna setting of our work to the case when the delivery phase is error prone as well as to the case where some users can request the same files. The work [57] considered a shared-cache setting where each group of two users can be connected in four different ways to two helper nodes/caches. While we focused on uncoded cache placement, coded cache placement was considered in the work in [58]. In [59], it was shown that our derived optimal performance can be improved if the server has full knowledge — during cache placement — of the number of users associated to each cache. While in [59] the authors considered the setting where all the caches have the same size, which cannot be optimized, in the next chapter we will show that the knowledge of the network topology turns out to be much more impactful if we can allocate the memory across the caches as a function of the number of users connected to them. A decentralized scheme for the shared-link shared-cache network was introduced in [60]. Another interesting work is the one in [61] where the authors provided statistical analysis of the average performance of the shared-link shared-cache networks studied in this chapter, and where simple load balancing techniques are proposed to combat the performance loss due to the heterogeneity of the number of users associated to each cache.

Chapter 4

Topology-Aware Shared-Cache Networks

This chapter studies a shared-cache setting similar to the one in the previous chapter, but where now the number of users assisted by each cache is known during the cache placement phase. Unlike the previous chapter, we will only consider the single shared-link setting. However, in light of the results in the previous chapter, the techniques developed for this shared-link setting can be extended to the case where the transmitter is equipped with multiple antennas. For such cache-aided shared-link setting, we will show how a proper allocation of a total cache memory of t times the library size across the caches, as a function of their occupancies, allows to always achieve a sum DoF of $t+1$ in conjuction with a local caching gain that surprisingly increases with the skewness of the cache occupancy vector $\mathbf{L}$. This reveals the importance of memory allocation in heterogeneous scenarios, which allows for a significantly improved performance over topology-agnostic schemes. Under some basic assumptions, we will prove the optimality of the proposed achievable scheme by developing a novel converse bound based on index coding that captures the heterogeneity of the setting. Finally, motivated by the fact that the topology of the network might vary over time, we will also address a topology-partially-aware scenario where the cache placement is designed according to a topology that does not match with the network topology in the upcoming delivery phase. The scheme proposed for this scenario allows for flexibility in handling different network topologies during the delivery phase while still benefiting of the gains coming from memory allocation.

4.1 System Model and Problem Formulation

In this section, we present the system model with two different operating scenarios as well as the corresponding performance metric. The system model will be fully described for the topology-aware scenario, while the description of the topology-partially-aware scenario is limited to highlight the differences with the previous scenario. Hereinafter, whenever it is not explicitly mentioned, we will always refer to the topology-aware scenario.

Topology-Aware Scenario

Similarly as in the previous chapter[1], we consider a cache-aided network where a transmitter (TX) with access to a library of N unit-sized files $W^{(1)}, W^{(2)}, \dots, W^{(N)}$ is connected via a shared-link broadcast channel to K users ($N \geq K$), each of which is connected to *one* of Λ different caches. The size of each cache $\lambda \in \{1, 2, \dots, \Lambda\}$ is a design parameter denoted by $M_\lambda \in (0, N]$ (in units of file), adhering to a cumulative sum cache-size constraint $M_\Sigma \triangleq \sum_{\lambda=1}^{\Lambda} M_\lambda$. We define the normalized cache size of cache λ as $\gamma_\lambda \triangleq \frac{M_\lambda}{N}$, such that the sum cache-size constraint takes the form

$$\sum_{\lambda=1}^{\Lambda} \gamma_\lambda = \frac{M_\Sigma}{N} \triangleq t. \tag{4.1}$$

We consider a normalized scenario where the channel capacity is normalized to one file per unit of time, and we assume that users can access the content of the cache to which they are connected at zero cost. Each cache λ is connected to a set of users $\mathcal{U}_\lambda \subset [K]$, such that all these disjoints sets $\mathcal{U} = \{\mathcal{U}_1, \dots, \mathcal{U}_\Lambda\}$ form a partition of the set of users $[K]$. The number of users connected to cache λ is denoted by L_λ (i.e. $|\mathcal{U}_\lambda| = L_\lambda$) for any $\lambda \in [\Lambda]$, and it is here referred to as *occupancy* of cache λ. This defines a so-called *cache occupancy* vector

$$\mathbf{L} \triangleq (L_1, \dots, L_\Lambda), \tag{4.2}$$

where naturally $\sum_{\lambda=1}^{\Lambda} L_\lambda = K$, and where we assume, without loss of generality, that $L_i \geq L_j$ for $i < j$, such that $L_1 \geq L_2 \geq \cdots \geq L_\Lambda$. Whenever needed, we will use $\mathbf{L}$ as a set, i.e. $\mathbf{L} = \{L_\lambda\}_{\lambda=1}^{\Lambda}$. The schematic of this setting is depicted in Figure 4.1.

Remark 4. *It is important to highlight that here we have assumed that cache λ is the λ-th most populated cache such that $L_\lambda = |\mathcal{U}_\lambda|$, in contrast to the previous chapter where cache λ was not necessarily the λ-th most populated cache.*

The system works in three different phases.

1. A *memory allocation phase* during which the knowledge of the cache occupancy vector $\mathbf{L}$ is used to allocate the total memory M_Σ to the caches, yielding the allocated size set $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$.

2. A *cache placement phase* during which each cache λ — of allocated size γ_λ — is filled with content $\mathcal{Z}_\lambda$ from the library, according to a certain strategy[2] $\boldsymbol{Z} = (\mathcal{Z}_1, \dots, \mathcal{Z}_\Lambda)$. In this work, we focus only on *uncoded cache placement schemes* (cf. [5]), where each cache stores part of the content of the library as it is, without any coding. Besides this, we also assume that the cache occupancy vector $\mathbf{L}$ is known in this phase.

[1] Some of the definitions and assumptions in this section coincide with the ones of the system model in the previous chapter. This allows the chapter to be self-contained.

[2] We notice that the size of cache λ, i.e. $|\mathcal{Z}_\lambda|$, reflects the memory allocation of the previous phase.

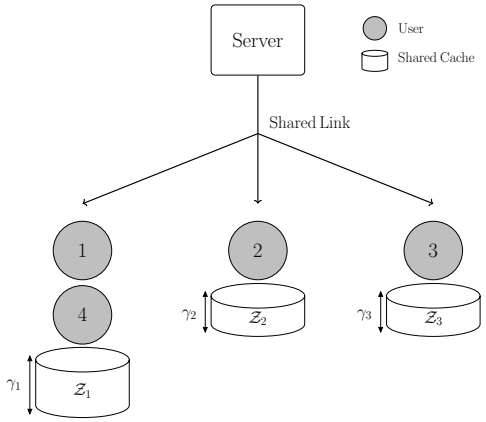


Figure 4.1 – Schematic of the shared-cache shared-link setting with optimized cache sizes for $\mathbf{L} = (2, 1, 1)$.

3. A *delivery phase* that starts with each user $k \in [K]$ requesting a file. The vector of requested file indices is denoted by $\mathbf{d} \triangleq (d_1, d_2, \ldots, d_K)$, such that the file requested by user k is given by $W^{(d_k)}$. Once the demand vector $\mathbf{d}$ is known, the server will aim to deliver each requested file to its corresponding user.

Topology-Partially-Aware Scenario

In this scenario, we consider a similar setting to the one described above with the only difference that the number of users that will be connected to each cache, i.e. $\mathbf{L}$, is not known during the memory allocation and cache placement phases. Nevertheless, we assume that the first two phases are designed according to another cache occupancy vector vector $\bar{\mathbf{L}}$. For any $\lambda \in [\Lambda]$, the value of $\bar{L}_\lambda$ can represent — but it is not limited to — the expected number of users connected to cache λ during the delivery phase. We also allow the sum $\sum_{\lambda=1}^{\Lambda} \bar{L}_\lambda$ to be arbitrary and not necessarily equal to $\sum_{\lambda=1}^{\Lambda} L_\lambda = K$, while at the same time we assume that for any $\lambda \in [\Lambda]$ the value of L_λ is strictly positive.

4.1.1 Problem Definition

We consider the standard *rate* metric that has been commonly used in coded caching literature [2, 4, 5], to which we hereinafter refer to as *delivery time* and whose definition follows. For any given *uncoded cache placement* scheme $\boldsymbol{Z}$, any cache occupancy vector $\mathbf{L}$ and any given demand $\mathbf{d}$, we define $T^*(\boldsymbol{Z}, \mathbf{d}, \mathbf{L})$ as the minimum delivery time (over all delivery schemes) that guarantees delivery of the desired files $W^{(d_k)}$ to all users $k \in [K]$. For the topology-aware scenario, under the assumption of uncoded cache placement, our goal is to characterize the minimum worst-case delivery time over all memory-allocation strategies and all placement-and-delivery schemes, i.e.,

$$T^*(t, \mathbf{L}) \triangleq \min_{\boldsymbol{Z}} \max_{\mathbf{d}} T^*(\boldsymbol{Z}, \mathbf{d}, \mathbf{L}) \tag{4.3}$$

as a function of t and $\mathbf{L}$. We omit hereinafter the dependence of T^* on $(t, \mathbf{L})$ when there is not any possible ambiguity. We also define the optimal cache placement $\boldsymbol{Z}^*_{\mathbf{L}}$ as

$\boldsymbol{Z}^*_{\mathbf{L}} \triangleq \underset{\boldsymbol{Z}}{\operatorname{argmin}} \max_{\mathbf{d}} T^*(\boldsymbol{Z}, \mathbf{d}, \mathbf{L})$. Next, we present the definition of homogeneous cache placement.

Definition 2. A cache placement scheme is *homogeneous* if each bit of the library is repeated the same number of times throughout the different caches.

A homogeneous cache placement naturally implies that t must be an integer, i.e., $t \in [\Lambda]$. In the context of non-uniform $\mathbf{L}$ and heterogeneous $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$, the concept of the sum-DoF in cache-aided networks [62] naturally generalizes to

$$\text{DoF} \triangleq \frac{K - \sum_{\lambda=1}^{\Lambda} \gamma_\lambda L_\lambda}{T},$$

reflecting the rate of delivery of the non-cached desired information.

For the topology-partially-aware scenario, we will characterize the optimal delivery time

$$T^*(t, \mathbf{L}, \bar{\mathbf{L}}) \triangleq \max_{\mathbf{d}} T^*(\boldsymbol{Z}^*_{\bar{\mathbf{L}}}, \mathbf{d}, \mathbf{L}) \tag{4.4}$$

as a function of t, $\mathbf{L}$ and $\bar{\mathbf{L}}$.

4.2 An Illustrative Example of the Scheme for the Topology-Aware Setting

Before introducing our main results, we present in the following an illustrative example that shows the idea behind the proposed general scheme.

Consider an instance of the considered shared-cache problem with $N = 6$ files, $\Lambda = 3$ caches and $K = 6$ users distributed among the caches according to the cache occupancy vector $\mathbf{L} = (3, 2, 1)$. We assume that a sum memory of $M_\Sigma = 12$ units of file is available to be allocated across the caches.

In the caching phase, we split each file $W^{(n)}$ ($n \in [6]$) into 11 equally-sized subfiles indexed by the pairs[3]

$$\{(12,1),(12,2),(12,3),(12,4),(12,5),(12,6),(13,1),(13,2),(13,3),(23,1),(23,2)\}. \tag{4.5}$$

In the above, the first index of the pairs refers to the set of caches that will store the subfiles with that index. On the other end, the second index is just a counter that differentiates subfiles with the same first index. Then, for each $n \in [6]$, each cache λ stores those subfiles whose first index includes λ. It follows that the content of each cache is:

$$\mathcal{Z}_1 = \{W^{(n)}_{12,1}, W^{(n)}_{12,2}, W^{(n)}_{12,3}, W^{(n)}_{12,4}, W^{(n)}_{12,5}, W^{(n)}_{12,6}, W^{(n)}_{13,1}, W^{(n)}_{13,2}, W^{(n)}_{13,3} : n \in [6]\},$$

[3]For the sake of readability and concision, in (4.5) and throughout this example we have used the compact notation $(12, 1)$ in place of $(\{1, 2\}, 1)$. The same applies to all other indices.

$$\mathcal{Z}_2 = \{W_{12,1}^{(n)}, W_{12,2}^{(n)}, W_{12,3}^{(n)}, W_{12,4}^{(n)}, W_{12,5}^{(n)}, W_{12,6}^{(n)}, W_{23,1}^{(n)}, W_{23,2}^{(n)} : n \in [6]\},$$
$$\mathcal{Z}_3 = \{W_{13,1}^{(n)}, W_{13,2}^{(n)}, W_{13,3}^{(n)}, W_{23,1}^{(n)}, W_{23,2}^{(n)} : n \in [6]\},$$

thus adhering to $\gamma_1 = \frac{9}{11}, \gamma_2 = \frac{8}{11}, \gamma_3 = \frac{5}{11}$.

In the delivery phase, we consider the demand vector $\mathbf{d} = (1, 2, 3, 4, 5, 6)$ where $W^{(1)}, W^{(4)}$ and $W^{(6)}$ are each requested by one of the three users associated to cache 1, $W^{(2)}$ and $W^{(5)}$ by the two users with cache 2, and $W^{(3)}$ by the user with cache 3. For the sake of a more understandable exposition, we re-denote the ordered set of files $\{W^{(n)}\}_{n=1}^{6}$ as $\{A, B, C, D, E, F\}$. On that account, for $\mathcal{R}_\lambda$ denoting the set of uncached subfiles wanted by the users of cache λ, we have

$\mathcal{R}_1$	$\mathcal{R}_2$	$\mathcal{R}_3$
$A_{23,1}$	$B_{13,1}$	$C_{12,1}$
$D_{23,1}$	$E_{13,1}$	$C_{12,2}$
$F_{23,1}$	$B_{13,2}$	$C_{12,3}$
$A_{23,2}$	$E_{13,2}$	$C_{12,4}$
$D_{23,2}$	$B_{13,3}$	$C_{12,5}$
$F_{23,2}$	$E_{13,3}$	$C_{12,6}$

We notice that, thanks to the proposed heterogeneous memory allocation — which we will present in detail in Section 4.4 —, the number of subfiles (which are all equally-sized) requested from each cache is the same. This fact is key to allow the transmission of all data during the delivery phase in the form of multicast messages of order $t + 1 = 3$. Hence, the set of 6 XORs are

$$X_{123}(1) = D_{23,1} \oplus B_{13,1} \oplus C_{12,1} \tag{4.6}$$
$$X_{123}(2) = A_{23,1} \oplus E_{13,1} \oplus C_{12,2} \tag{4.7}$$
$$X_{123}(3) = A_{23,2} \oplus B_{13,2} \oplus C_{12,3} \tag{4.8}$$
$$X_{123}(4) = F_{23,1} \oplus E_{13,2} \oplus C_{12,4} \tag{4.9}$$
$$X_{123}(5) = F_{23,2} \oplus B_{13,3} \oplus C_{12,5} \tag{4.10}$$
$$X_{123}(6) = D_{23,2} \oplus E_{13,3} \oplus C_{12,6} \tag{4.11}$$

and can be easily decoded in the classical manner as in [2]. Consequently, the total delay is $T = \frac{6}{11}$ which can be shown to be exactly optimal under the assumption of uncoded and homogeneous cache placement. On the other hand, if the topology of the network was not known so that we should employ equal-sized caches, the best possible performance under the same assumptions is $T = 1$ (cf. Theorem 1), which almost doubles the delay of the new scheme. This gain of the new proposed scheme comes from the fact that the memory was allocated in a way that simultaneously increases the coded multicasting gain and the local caching gain. Intuitively, allocating more memory to those caches that serve more users automatically implies a higher local caching gain. At the same time, a cache allocation that levels out the data requested from each cache allowed for the design of multicast messages of maximum order. In contrast, in the case of uniform allocation

(see Chapter 3), a coding gain of $t+1$ is not achievable for all requested data, and the local caching gain is always smaller than any memory allocation strategy allocating more storage to more loaded caches.

4.3 Main Results

We present in this section our main contributions. We first start with an important definition that we will use throughout this chapter and Chapter 8.

Definition 3 (Elementary symmetric functions [63])**.** For any set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ of n elements, the k-th *elementary symmetric function* is defined as the sum of all distinct products of k distinct elements in $\mathcal{X}$, i.e.,

$$e_k(\mathcal{X}) \triangleq \sum_{q \in C_k^{\mathcal{X}}} \prod_{j=1}^{k} x_{q(j)}, \tag{4.12}$$

for any $k \in \{1, \ldots, n\}$, whereas $e_0(\mathcal{X}) \triangleq 1$.

We know proceed by presenting a lower bound on the delivery time under uncoded cache placement for the topology-aware scenario described before.

Topology-Aware Scenario

Theorem 4. *Under the assumption of uncoded cache placement, the optimal normalize delivery time of the* $(t, \mathbf{L})$ *shared-cache BC network satisfies*

$$T^*(t, \mathbf{L}) \geq \mathcal{T}_{low}^{(\bar{t}, \mathbf{L})}(t) \tag{4.13}$$

where $\bar{t} \triangleq \text{round}(t)$*,* $\mathcal{T}_{low}^{(\bar{t}, \mathbf{L})}(x)$ *is defined*[4] *as*

$$\mathcal{T}_{low}^{(\bar{t}, \mathbf{L})}(x) \triangleq \text{Conv}_{j \in [\Lambda]_0} \left(\frac{\sum_{q \in C_{\bar{t}+1}^{[\Lambda]}} \frac{\bar{t}+1-|q \cap \tau_j^\star|}{j+1-|q \cap \tau_j^\star|} \prod_{i=1}^{\bar{t}+1} L_{q(i)}}{\sum_{\ell \in C_{\bar{t}}^{[\Lambda]}} \prod_{i=1}^{\bar{t}} L_{\ell(i)}} \right), \tag{4.14}$$

and $\tau_j^\star$ *is given by*

$$\tau_j^\star = \begin{cases} \{\emptyset\} & \text{if } j = 0 \\ \{\Lambda - j + 1,\ \Lambda - j + 2,\ \ldots,\ \Lambda\} & \text{if } 1 \leq j < \bar{t} \\ \{1,\ 2,\ \ldots,\ j\} & \text{if } j \geq \bar{t} \end{cases}$$

Proof. The proof is presented in Section 4.5. □

[4] As reported in the notation section at the beginning of this manuscript, the term C_k^χ is used to denote the set of all k-combinations of the set χ.

The heterogeneity of the cache occupancy vector $\mathbf{L}$ and its knowledge during the placement phase introduce a new challenge in the derivation of the converse bound. Directly adjusting and applying the index coding technique developed in [4] (or, equivalently, the genie-aided approach of [5]) would result in a very loose bound, as we have demonstrated[5] in [43]. On the other hand, the derivation of the bound in (4.14) relies on a new index-coding-based approach that captures the heterogeneity of the system and therefore — in conjunction with the developed achievable coded caching scheme presented in Section 4.4 — allows us to characterize the optimal delivery time under the basic assumptions of homogeneous (cf. definition 2) and uncoded cache placement in the subsequent theorem. We first present the achievable delivery time of the scheme in Section 4.4 in the following lemma.

Lemma 3. *For the K-user BC with Λ shared caches, normalized sum-cache constraint t, and cache occupancy vector $\mathbf{L}$, the worst-case delivery time*

$$T(t,\mathbf{L}) = \mathrm{Conv}_{t\in[\Lambda]_0}\left(\frac{\sum_{\lambda=1}^{\Lambda} L_\lambda(1-\gamma_\lambda)}{t+1}\right) \tag{4.15}$$

is achievable, where the memory allocation $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$ is given by

$$\gamma_\lambda = \frac{L_\lambda \cdot e_{t-1}(\mathbf{L}\setminus\{L_\lambda\})}{e_t(\mathbf{L})}. \tag{4.16}$$

Proof. The placement and delivery schemes are presented in Section 4.4. □

We immediately notice that the delay in (4.15) directly implies that $\mathrm{DoF} = t+1$, which is an improvement over the topology-agnostic scenario discussed in Chapter 3. In fact, we know from Chapter 3 that, if during the cache placement phase the exact number of users that will be connected to each cache during the delivery phase is not known, then $\mathrm{DoF} = t+1$ can be achieved only in the uniform case where we have $\frac{K}{\Lambda}$ users per cache, and that any non-uniformity in $\mathbf{L}$ forces a DoF penalty. The above lemma shows that knowledge of the cache occupancy vector $\mathbf{L}$ allows for a redesigned and skewed memory allocation that, in turn, simultaneously allows a better local caching gain and a higher sum-DoF. To clarify this, a strategy that allocates more memory to more loaded caches automatically allows for higher local caching gains than a uniform memory allocation across the caches. At the same time, such heterogeneous allocation allows for multicasting messages that always serve $t+1$ users at a time. These observations lead to the surprising fact that, for a fixed number of users and caches, the uniform cache occupancy vector $\mathbf{L} = (\frac{K}{\Lambda}, \frac{K}{\Lambda}, \ldots, \frac{K}{\Lambda})$ is the one that results in the highest delivery time, while the more skewed is the cache occupancy vector $\mathbf{L}$ the lower is the achievable normalized delivery time. Also, it is interesting to observe that the skewness of the allocated $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$ reduces as t increases. For $t=1$, the skewness is maximal since we have $\gamma_\lambda = \frac{L_\lambda}{K}$, while as t increases, the skewness reduces down to having a uniform allocation when $t=\Lambda$.

[5]The converse bound presented in [43] is not included in this thesis since the bound presented in Section 4.5 improves the one in [43].

Observation 2. *It is interesting to observe that, for any given normalized total cache size t, the memory allocation $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$ given in Lemma 3 coincides with the one of the scheme proposed in [38] (see also [37]) for a cache-aided setting with fixed unequal channel capacities. Precisely, Tang et al. consider a wireless broadcast channel where the link between the transmitter and cache-aided receiver λ has normalized capacity R_λ, and where these capacities are known during the cache allocation and placement phases. Assuming that each user requests only one file, the authors proposed a scheme which requires an unequal cache size allocation that coincides with the one in* (4.16)*, where L_λ would be replaced by the inverse of the rate of user λ, i.e. $R_\lambda = \frac{1}{L_\lambda}, \forall \lambda \in [\Lambda]$. Despite the different nature of these two problems, we can observe that in the unequal link rate setting a user with rate R_i (here referred to as user i) is reminiscent of the user requesting L_i files (or the cache serving L_i users) in our setting.*

We have presented the proposed lower bound and achievable scheme. In the next theorem, we state the conditions under which the achievable delivery time in (4.15) is exactly optimal.

Theorem 5. *For integer values of $t \in [\Lambda]_0$, the achievable delivery time $T(t, \mathbf{L})$ in* (4.15) *is exactly optimal in the following two cases:*

(i) *Under the assumption of uncoded and homogeneous cache placement.*

(ii) *Under the assumption of uncoded cache placement when the sequence $\{\tilde{c}^{(t)}_{\tau^\star_j}\}_{j\in[\Lambda]_0}$ is convex in j, where*

$$\tilde{c}^{(t)}_{\tau^\star_j} \triangleq \frac{\sum_{q\in C^{[\Lambda]}_{t+1}} \frac{t+1-|q\cap\tau^\star_j|}{j+1-|q\cap\tau^\star_j|} \prod_{i=1}^{t+1} L_{q(i)}}{\sum_{\ell\in C^{[\Lambda]}_t} \prod_{i=1}^{t+1} L_{\ell(i)}}. \tag{4.17}$$

Proof. The converse bound for (i) is relegated to Section 4.5. We prove (ii) in the following. From (4.17), we can write $\mathcal{T}^{(t,\mathbf{L})}_{low}(x)$ (defined in (4.14)) as

$$\mathcal{T}^{(t,\mathbf{L})}_{low}(x) = \mathrm{Conv}_{j\in[\Lambda]_0}\left(\tilde{c}^{(t)}_{\tau^\star_j}\right). \tag{4.18}$$

For any convex sequence, its lower convex envelope contains all its elements. Thus, if $\{\tilde{c}^{(t)}_{\tau^\star_j}\}_{j\in[\Lambda]_0}$ is convex, then $\mathcal{T}^{(t,\mathbf{L})}_{low}(t) = \tilde{c}^{(t)}_{\tau^\star_t}$ for any integer point $t \in [\Lambda]_0$.

Moreover, for $j = t$, we have that $\frac{t+1-|q\cap\tau^\star_j|}{j+1-|q\cap\tau^\star_j|} = 1$ for any q, and thus (4.17) yields $\tilde{c}^{(t)}_{\tau^\star_t} = \frac{\sum_{q\in C^{[\Lambda]}_{t+1}} \prod_{i=1}^{t+1} L_{q(i)}}{\sum_{\ell\in C^{[\Lambda]}_t} \prod_{i=1}^{t+1} L_{\ell(i)}} = \frac{e_{t+1}(\mathbf{L})}{e_t(\mathbf{L})}$. Then, it follows from Theorem 4 that

$$\begin{aligned} T^*(t,\mathbf{L}) &\geq \frac{e_{t+1}(\mathbf{L})}{e_t(\mathbf{L})} \\ &\overset{(a)}{=} \frac{1}{t+1} \frac{\sum_{\lambda=1}^{\Lambda} L_\lambda \cdot e_t(\mathbf{L}\setminus\{L_\lambda\})}{e_t(\mathbf{L})} \end{aligned}$$

$$\begin{aligned} &\overset{(b)}{=} \frac{1}{t+1} \frac{\sum_{\lambda=1}^{\Lambda} L_\lambda \cdot (e_t(\mathbf{L}) - L_\lambda e_{t-1}\mathbf{L} \setminus \{L_\lambda\})}{e_t(\mathbf{L})} \\ &\overset{(c)}{=} \frac{\sum_{\lambda=1}^{\Lambda} L_\lambda(1-\gamma_\lambda)}{t+1} = T(t,\mathbf{L}), \end{aligned} \tag{4.19}$$

where (a) follows from using Corollary 3, (b) follows from employing Property 1 in Appendix B.1 and (c) is because of (4.16). □

Next, we present an example of a setting for which our scheme is optimal under the constraint of uncoded cache placement.

Example 1. *Consider the cache-aided network of the example in Section 4.2 with* $\mathbf{L} = (3,2,1)$ *and* $t = 2$. *The sequence* $\left\{\tilde{c}^{(2)}_{\tau^\star_j}\right\}_{j\in\{0,1,2,3\}}$ *takes the values* $\{18/11, 12/11, 6/11, 0\}$, *which is a convex sequence. From Theorem 5, this implies that for the considered example in Section 4.2 the achievable delivery time* $T^*(2,(3,2,1)) = 6/11$ *is information-theoretic optimal under the assumption of uncoded cache placement.*

Remark 5. *It is important to highlight that the converse bound in Theorem 4 and the achievable delivery time in Lemma 3 can be extended to the setting where the server is equipped with* N_0 *antennas. In particular, as long as each cache assists at least* N_0 *users, i.e.* $L_\lambda \geq N_0$, $\forall \lambda \in [\Lambda]$, *the achievable delivery time reduces by a multiplicative factor of* N_0, *similarly as it happened in the topology-agnostic setting discussed in the previous chapter. The main idea behind the multi-antenna version of the proposed scheme in Section 4.4 is to increase the subpacketization of each file by a multiplicative factor of* N_0, *thus allowing to use the* N_0 *antennas for multiplexing users with the same cache, as we saw in the previous chapter. A valid matching lower bound can be also easily obtained by dividing the developed bound in Theorem 4 by a factor* N_0, *again following the same line as in Section 3.4 of the previous chapter.*

Topology-Partially-Aware Scenario

We now shift the focus to the topology-partially-aware scenario, for which we present the optimal delivery time in the next theorem. Hereinafter, we will use $S_{\Lambda,\Lambda-t}$ to denote the set of the $(\Lambda-t)$-permutations of $[\Lambda]$.

Theorem 6. *For the* $(t, \bar{\mathbf{L}}, \mathbf{L})$ *topology-partially-aware scenario, the delivery time*

$$T^*(t,\bar{\mathbf{L}},\mathbf{L}) = \max_{\sigma\in S_{\Lambda,\Lambda-t}} \sum_{\lambda=1}^{\Lambda-t} L_{\sigma(\lambda)} \sum_{q\in C_t^{[\Lambda]\setminus\{\sigma(1),\ldots,\sigma(\lambda)\}}} \frac{\dot{\bar{L}}_q}{e_t(\bar{\mathbf{L}})} \tag{4.20}$$

is exactly optimal under the assumption of placement $\boldsymbol{Z}^*_{\bar{\mathbf{L}}}$.

Theorem 6 describes the optimal delivery time for a scenario in which, during the memory allocation and cache placement phases, the server has a wrong (e.g., noisy or statistical) information about the cache occupancy vector of the subsequent delivery phase. Equivalently, during the first two phases the server knows that users will be

connected to the caches in the delivery phase according to cache occupancy vector $\bar{\mathbf{L}}$, but it turns out that the actual cache occupancy vector will finally be $\mathbf{L}$. The delivery time in Theorem 6 is higher than the optimal delivery time for the scenario where users actually show up according to $\bar{\mathbf{L}}$, as expected by the server, i.e. $T^*(t, \bar{\mathbf{L}}, \mathbf{L}) \geq T(t, \bar{\mathbf{L}})$, where $T(t, \bar{\mathbf{L}})$ was presented in (4.15). Note that, in scenarios where only long-term statistical information is available, it is common to consider the strategy of acting as if this information was perfect and true for any realization. In Section 4.7, we will show through some numerical evaluations how $T^*(t, \bar{\mathbf{L}}, \mathbf{L})$ behaves on average when $\mathbf{L}$ is a realization of Λ independent Poisson random variables.

4.4 Achievability for the Topology-Aware Scenario

In this section, we present our caching and delivery scheme that achieves the optimal delivery time in (4.15), followed by the assessment of its performance. The scheme below is presented for integer values of t, i.e., $t \in \{1, 2 \ldots, \Lambda\}$; for all other cases ($t \notin \mathbb{N}$), memory-sharing is used (cf. [2]), and we address these cases at the end of this section.

4.4.1 Memory Allocation and Cache Placement

In the first place, we split each file $W^{(n)}, n \in [N]$, into

$$S = e_t(\mathbf{L}) = \sum_{\tau \in C_t^{[\Lambda]}} \prod_{j=1}^{t} L_{\tau(j)}, \tag{4.21}$$

subfiles of equal size, such that $W^{(n)}$ is partitioned as

$$W^{(n)} = \left\{ W_{\tau,1}^{(n)}, W_{\tau,2}^{(n)}, \ldots, W_{\tau,|A_\tau|}^{(n)} \mid \tau \in C_t^{[\Lambda]} \right\}$$

where $A_\tau \triangleq \left\{ 1, 2, \ldots, \prod_{j=1}^{t} L_{\tau(j)} \right\}$. Afterwards, cache $\lambda \in [\Lambda]$ stores in its memory all subfiles $W_{\tau,m_\tau}^{(n)}, m_\tau \in A_\tau$, whose first subscript τ includes λ, which in turn results in the following cache

$$\mathcal{Z}_\lambda = \left\{ W_{\tau,m_\tau}^{(n)} \mid W_{\tau,m_\tau}^{(n)} \in W^{(n)}, \tau \ni \lambda, m_\tau \in A_\tau, n \in [N] \right\}.$$

This automatically yields the memory allocation

$$\gamma_\lambda = \frac{L_\lambda \cdot e_{t-1}(\mathbf{L} \setminus \{L_\lambda\})}{e_t(\mathbf{L})}, \quad \lambda \in [\Lambda]. \tag{4.22}$$

Proof of equation (4.22)*:* To evaluate γ_λ, we first note that all subfiles are equally-sized and that the placement scheme is *symmetric* with respect to the library files, i.e. the caching strategy does not depend on the file index $n \in [N]$. This suggests that γ_λ can be evaluated as the ratio between the number of subfiles (of any file $W^{(n)}$) stored in

cache λ and the total number of subfiles $S = e_t(\mathbf{L})$ into which $W^{(n)}$ is split. The fact that a subfile $W^{(n)}_{\tau,m}$ is placed in $\mathcal{Z}_\lambda$ if and only if $\lambda \in \tau$, together with the fact that $|A_\tau| = \prod_{j=1}^{t} L_{\tau(j)}$, automatically yield the numerator of (4.22). □

This same placement also assures that each subfile is cached in exactly t caches (because each τ satisfies $|\tau| = t$), which in turn guarantees the sum memory constraint in (4.1). This memory constraint can also be verified by noting that

$$\sum_{\lambda=1}^{\Lambda} \gamma_\lambda = \sum_{\lambda=1}^{\Lambda} \frac{L_\lambda \cdot e_{t-1}(\mathbf{L} \setminus \{L_\lambda\})}{e_t(\mathbf{L})} = t, \tag{4.23}$$

where the last step follows directly from Corollary 3 in Appendix B.1. Also, this placement yields an interesting property — described in the following proposition — which will be instrumental for the design of the delivery phase and the achieving of its performance.

Proposition 1. *For any $(t+1)$-tuple $\mathcal{Q} \subset [\Lambda]$, and for any specific cache $\lambda \in \mathcal{Q}$, the total number of subfiles with first subscript $\tau = \mathcal{Q} \setminus \{\lambda\}$ that are missing from all the users associated to cache λ is independent of λ and it equals*

$$P_{\mathcal{Q}} \triangleq \prod_{j=1}^{t+1} L_{\mathcal{Q}(j)}. \tag{4.24}$$

Proof. For any $(t+1)$-tuple $\mathcal{Q} \subset [\Lambda]$, consider cache $\lambda \in \mathcal{Q}$ and let $\tau = \mathcal{Q} \setminus \{\lambda\}$. There are L_λ requested files from the users $\mathcal{U}_\lambda$ of cache λ, each having $\prod_{j=1}^{t} L_{\tau(j)}$ subfiles with first index τ. This means that the total number of subfiles that need to be sent to serve users in $\mathcal{U}_\lambda$ is $L_\lambda \prod_{j=1}^{t} L_{\tau(j)} = \prod_{j=1}^{t+1} L_{\mathcal{Q}(j)}$, which does not depend on λ. □

4.4.2 Delivery Scheme

For ease of presentation, we will use $\mathbf{d}_\lambda$ to denote the vector of indices of the files requested by the users in $\mathcal{U}_\lambda$. For a fixed $(t+1)$-tuple $\mathcal{Q}$ and any $\lambda \in \mathcal{Q}$, consider the set of subfiles

$$\{W^{(\mathbf{d}_\lambda(j))}_{\tau,m} : j \in [L_\lambda], m \in A_\tau\}$$

with first subscript $\tau = \mathcal{Q} \setminus \{\lambda\}$ that are requested from users in $\mathcal{U}_\lambda$. From Proposition 1, we know that the cardinality of this set is $P_{\mathcal{Q}}$, and thus we can relabel these subfiles as

$$\{F^{(\lambda)}_{\tau,j} : j \in [P_{\mathcal{Q}}]\}.$$

Because of the cache placement phase, we note that, for any $(t+1)$-tuple $\mathcal{Q}$ and any $j \in [P_{\mathcal{Q}}]$, the set of subfiles

$$F^{(\lambda)}_{\mathcal{Q}\setminus\{\lambda\},j}, \ \forall \lambda \in \mathcal{Q}, \tag{4.25}$$

forms a clique of $t+1$ nodes. By Proposition 1, for any $(t+1)$-tuple $\mathcal{Q} \in [\Lambda]$, we have $P_{\mathcal{Q}}$ cliques as in (4.25), all of $t+1$ nodes. Consequently, we transmit, for each $(t+1)$-tuple $\mathcal{Q} \subseteq [\Lambda]$, the following $P_{\mathcal{Q}}$ XORs:

$$X_{\mathcal{Q}}(j) = \bigoplus_{\lambda \in \mathcal{Q}} F^{(\lambda)}_{\mathcal{Q}\setminus\{\lambda\},j}, \quad \forall j \in [P_{\mathcal{Q}}], \tag{4.26}$$

whose structure allows for clique-based decoding as in [2].

4.4.3 Performance of the Scheme

The fact that there are $P_{\mathcal{Q}}$ XORs for each $(t+1)$-tuple Q implies a total of

$$\sum_{\mathcal{Q}\in C_{t+1}^{[\Lambda]}} P_{\mathcal{Q}} = \sum_{\mathcal{Q}\in C_{t+1}^{[\Lambda]}} \prod_{j=1}^{t+1} L_{\mathcal{Q}(j)} = e_{t+1}(\mathbf{L})$$

transmissions, and a corresponding delivery time of

$$T(t,\mathbf{L}) = \frac{e_{t+1}(\mathbf{L})}{e_t(\mathbf{L})}, \tag{4.27}$$

where the denominator $e_t(\mathbf{L})$ is due to (4.21). In the proof of Theorem 5, we have seen that the above achievable delivery time in (4.27) can be written in the more standard form

$$T(t,\mathbf{L}) = \frac{\sum_{\lambda=1}^{\Lambda} L_\lambda(1-\gamma_\lambda)}{t+1}, \tag{4.28}$$

where $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$ is the memory allocation obtained in (4.22).

Extension to Non-integer Values of t

When the normalized total memory t has a non-integer value, we apply memory sharing along the same lines as in [2]. We write t as $t = \alpha\lfloor t\rfloor + (1-\alpha)\lceil t\rceil$, for $\alpha \in [0,1]$, and we split each file $W^{(n)}$ of the library in two parts $W^{(n),1}, W^{(n),2}$, where $|W^{(n),1}| = \alpha$ and $|W^{(n),2}| = (1-\alpha)$, such that the library remains partitioned in two sub-libraries

$$\mathcal{W}_1 = \{W^{(n),1} | n \in [N]\}, \quad \mathcal{W}_2 = \{W^{(n),2} | n \in [N]\}.$$

Afterwards, we first employ the cache placement scheme in Section 4.4.1 for sub-library $\mathcal{W}_1$ with a total sum-cache constraint $\lfloor t\rfloor$, and then we do the same for sub-library $\mathcal{W}_2$ with a total sum-cache constraint $\lceil t\rceil$. Delivery phase now consists of 2 rounds, each as in Section 4.4.2: the first round employs XORs of order $\lfloor t\rfloor + 1$ to serve files $\{W^{(d_k),1} | k \in [K]\}$, whereas the second round employs XORs of order $\lceil t\rceil + 1$ to serve $\{W^{(d_k),2} | k \in [K]\}$. This scheme clearly results in the following delivery time

$$T(t,\mathbf{L}) = \alpha T(\lfloor t\rfloor, \mathbf{L}) + (1-\alpha) T(\lceil t\rceil, \mathbf{L}). \tag{4.29}$$

We now present a lemma that is instrumental for the proof of Lemma 3.

Lemma 4. *The sequence* $\left\{\frac{e_{t+1}(\mathbf{L})}{e_t(\mathbf{L})}\right\}_{t\in[\Lambda]_0}$ *is a decreasing and convex sequence.*

Proof. The proof is relegated to Appendix B.2. □

The achievability of (4.29) implies that, for integer t, the straight line between points $(t, T(t,\mathbf{L}))$ and $(t+1, T(t+1,\mathbf{L}))$ is also achievable. Moreover, we know from (4.27) that $\{T(t,\mathbf{L})\}_{t\in[\Lambda]_0} = \left\{\frac{e_{t+1}(\mathbf{L})}{e_t(\mathbf{L})}\right\}_{t\in[\Lambda]_0}$, and thus Lemma 4 implies that $\{T(t,\mathbf{L})\}_{t\in[\Lambda]_0}$ is a convex sequence. Since the lower convex envelope of a convex sequence is a piece-wise function composed of the segments connecting two successive elements of the sequence $\{T(t,\mathbf{L})\}_{t\in[\Lambda]_0}$ (i.e., (4.29)), Lemma 3 is proven.

4.5 Converse for the Topology-Aware Scenario

In this section, we present a converse bound on the optimal delivery time $T^*(t, \mathbf{L})$, which will serve as a proof for Theorem 4. We will also show that, restricting the cache placement scheme to be homogeneous as in Definition 2, the proposed lower bound naturally implies the optimality of the achievable delivery time in Lemma 3, thus proving case (i) of Theorem 5.

In what follows, we denote the set of demands with distinct file requests by $\mathcal{D}_{wc}$, such that $\mathcal{D}_{wc} \triangleq \{\mathbf{d} \in [N]^K : d_j \neq d_i,\ \forall\ i \neq j\}$. Finally, we will use the notation $W_{\tau}^{(i)}$ to refer to the part of file $W^{(i)}$ exclusively stored in the caches in set τ.

Lower bounding $T^*(\boldsymbol{Z}, \mathbf{d}, \mathbf{L})$

We first present a lemma that provides a *generic* lower bound on the delivery time $T^*(\boldsymbol{Z}, \mathbf{d}, \mathbf{L})$ as a function of the cache permutation $\sigma \in \mathcal{S}_\Lambda$.

Lemma 5. *Consider the delivery phase of a shared-cache network with a cache placement described by $\boldsymbol{Z}$, demand vector $\mathbf{d}$ and cache occupancy vector $\mathbf{L}$. Then, under the assumption of uncoded cache placement, the optimal delivery time $T^*(\boldsymbol{Z}, \mathbf{d}, \mathbf{L})$ can be lower bounded by the quantity*

$$T_{lb,\sigma}(\boldsymbol{Z}, \mathbf{d}, \mathbf{L}) \triangleq \sum_{\lambda=1}^{\Lambda} \sum_{\ell=1}^{L_{\sigma(\lambda)}} \sum_{\tau_\lambda \subseteq [\Lambda] \backslash \{\sigma(1),\ldots,\sigma(\lambda)\}} |W_{\tau_\lambda}^{(\mathbf{d}_{\sigma(\lambda)}(\ell))}|, \tag{4.30}$$

where σ denotes an arbitrary permutation of the set of caches $[\Lambda]$, i.e. $\sigma \in S_\Lambda$.

Proof. The lemma is direct from equation (3.44) in Section 3.4, employed here with $N_0 = 1$. Furthermore, equation (3.44) was stated for a specific permutation denoted by σ_s; it can be easily verified that it actually holds for every permutation $\sigma \in S_\Lambda$, as stated in the above lemma. □

Next, a *flexible* lower bound on $T^*(\boldsymbol{Z}, \mathbf{d}, \mathbf{L})$ can be constructed as a weighted average of the $\Lambda!$ possible lower bounds that stem from (4.30). Thus, it holds that

$$T^*(\boldsymbol{Z}, \mathbf{d}, \mathbf{L}) \geq \sum_{\sigma \in \mathcal{S}_\Lambda} w_\sigma T_{lb,\sigma}(\boldsymbol{Z}, \mathbf{d}, \mathbf{L}), \tag{4.31}$$

where the weights $\{w_\sigma\}$ satisfy $\sum_{\sigma \in \mathcal{S}_\Lambda} w_\sigma = 1$.

Remark 6. *The use of the weighted average is key to the construction of a tight bound, and it is one of the key contributions of our work. In fact, in many works that follow the index coding approach originally proposed in [4] (and the similar genie-aided approach used in [5]) to construct lower bounds on coded caching problems, it is common to lower bound the delivery time as a uniform average of several bounds driven by a certain cache permutation σ (see for example [5, 64, 65]). This method has been shown to work well in*

settings that are uniform in terms of number of users per cache and sizes of the caches. However, whenever the system model is affected by some heterogeneity, this approach can easily fail to meet the goal. In our particular setting, we have shown in [43] that the uniform average (i.e. $w_\sigma = \frac{1}{\Lambda!}$) leads to a loose bound which proved our achievable performance to be optimal within a gap that scales linearly with the normalized total cache size t. On the other hand, in this work, we show that a careful choice of the weights can be crucial to derive a tight bound. We believe that this approach can be helpful to derive lower bounds for other generic heterogeneous coded caching problems.

In our derivation, for any $p \in [\Lambda]_0$, the choice of the weights w_σ is taken as

$$w_\sigma^{(p)} \triangleq \frac{\prod_{j=1}^{p} L_{\sigma(\Lambda-p+j)}}{\sum_{\sigma\in\mathcal{S}_\Lambda} \prod_{j=1}^{p} L_{\sigma(\Lambda-p+j)}}, \tag{4.32}$$

where we have used the upper index $^{(p)}$ to highlight the dependency of the value of the weights on the choice of the parameter p. For $p = 0$ we define $w_\sigma^{(0)} \triangleq \frac{1}{|\mathcal{S}_\Lambda|}$.

Lower bound on $T^*(t, \mathbf{L})$

We now proceed to derive the lower bound on the optimal delivery time $T^*(t, \mathbf{L})$. In this respect, we start by bounding from below — for a fixed cache placement $\boldsymbol{Z}$ — the worst-case delay by the average rate over all demands with distinct requests as

$$T^*(\boldsymbol{Z}, \mathbf{L}) \triangleq \max_{\mathbf{d}} \; T^*(\boldsymbol{Z}, \mathbf{d}, \mathbf{L}) \geq \frac{1}{|\mathcal{D}_{wc}|} \sum_{\mathbf{d}\in\mathcal{D}_{wc}} T^*(\boldsymbol{Z}, \mathbf{d}, \mathbf{L}). \tag{4.33}$$

Combining (4.33) and (4.31), it yields

$$T^*(\boldsymbol{Z}, \mathbf{L}) \geq \frac{1}{|\mathcal{D}_{wc}|} \sum_{\mathbf{d}\in\mathcal{D}_{wc}} \sum_{\sigma\in\mathcal{S}_\Lambda} w_\sigma^{(p)} T_{lb,\sigma}(\boldsymbol{Z}, \mathbf{d}, \mathbf{L}) \tag{4.34}$$

$$\overset{(a)}{\geq} \underbrace{\frac{1}{|\mathcal{D}_{wc}|} \sum_{\mathbf{d}\in\mathcal{D}_{wc}} \sum_{\sigma\in\mathcal{S}_\Lambda} w_\sigma^{(p)} \sum_{\lambda=1}^{\Lambda} \sum_{\ell=1}^{L_{\sigma(\lambda)}} \sum_{\tau_\lambda\subseteq[\Lambda]\setminus\{\sigma(1),\ldots,\sigma(\lambda)\}} |W_{\tau_\lambda}^{(\mathbf{d}_{\sigma(\lambda)}(\ell))}|}_{\triangleq T_{lb,p}(\boldsymbol{Z},\mathbf{L})}, \tag{4.35}$$

where in (a) we have used (4.30).

Next, we write the R.H.S of (4.35), which we denote by $T_{lb,p}(\boldsymbol{Z}, \mathbf{L})$, in the more compact form

$$T_{lb,p}(\boldsymbol{Z}, \mathbf{L}) = \sum_{n=1}^{N} \sum_{\tau\in 2^{[\Lambda]}} c_{\tau,n}^{(p)} \frac{|W_\tau^{(n)}|}{N}, \tag{4.36}$$

where the value of $c_{\tau,n}^{(p)}$ is expressed in the following lemma. Before presenting the lemma, let us introduce the notation $\dot{\boldsymbol{L}}_q \triangleq \prod_{j=1}^{|q|} L_{q(j)}$ for any subset $q \subseteq [\Lambda]$ for the sake of readability.

Lemma 6. *The value of $c_{\tau,n}^{(p)}$ does not depend on the file index n and it takes the form*

$$c_{\tau}^{(p)} = \frac{1}{\sum_{\ell \in C_p^{[\Lambda]}} \dot{\boldsymbol{L}}_\ell} \left(\sum_{q \in C_{p+1}^{[\Lambda]}} \dot{\boldsymbol{L}}_q \frac{p+1-|q\cap\tau|}{|\tau|+1-|q\cap\tau|} + \frac{1}{|\tau|+1} \dot{\boldsymbol{L}}_\tau \sum_{s \in C_{p-j}^{[\Lambda]\setminus\{\tau\}}} \left(\dot{\boldsymbol{L}}_s \cdot \sum_{i=1}^{p-j} L_{s(i)} \right) \right). \tag{4.37}$$

Proof. The proof of this lemma is presented in Appendix B.3. □

We can tighten the bound on $T^*(\boldsymbol{Z}, \mathbf{L})$ by selecting the most restricting p, such that

$$T^*(\boldsymbol{Z}, \mathbf{L}) \geq \max_{p \in [\Lambda]_0} T_{lb,p}(\boldsymbol{Z}, \mathbf{L}). \tag{4.38}$$

From the definition of the optimal delay $T^*(t, \mathbf{L})$ in (4.3), using (4.38) and substituting (4.37) in (4.36) yields

$$\begin{aligned} T^*(t, \mathbf{L}) &= \min_{\boldsymbol{Z}} T^*(\boldsymbol{Z}, \mathbf{L}) && (4.39)\\ &\geq \min_{\boldsymbol{Z}} \max_{p \in [\Lambda]_0} \sum_{n=1}^{N} \sum_{\tau \in 2^{[\Lambda]}} c_{\tau}^{(p)} \frac{|W_\tau^n|}{N} \\ &= \min_{\boldsymbol{Z}} \max_{p \in [\Lambda]_0} \sum_{\tau \in 2^{[\Lambda]}} c_{\tau}^{(p)} a_\tau, && (4.40) \end{aligned}$$

where we have introduced the notation $a_\tau \triangleq \frac{1}{N}\left(|W_\tau^{(1)}| + |W_\tau^{(2)}| + ... + |W_\tau^{(N)}|\right)$. Now, considering the constraint on the total files size and the one on the sum cache size, a lower bound on the optimal delay $T^*(t, \mathbf{L})$ can be obtained from the solution of the following linear program

$$\begin{aligned} \min_{a_\tau} \max_{p \in [\Lambda]_0} & \sum_{\tau \in 2^{[\Lambda]}} c_{\tau}^{(p)} a_\tau \\ \text{subject to } & \sum_{\tau \in 2^{[\Lambda]}} a_\tau = 1, \\ & \sum_{\tau \in 2^{[\Lambda]}} |\tau| a_\tau = t, \\ & a_\tau \geq 0, \quad \forall \tau \in 2^{[\Lambda]}. \end{aligned} \tag{4.41}$$

Let us now focus on the proof of the lower bound in Theorem 4, and later we will consider the proof for the optimality in Theorem 5

Proof of Theorem 4

In what follows, we further bound from below the constructed lower bound in (4.41). First of all, let us introduce some useful notation. We define $\tilde{c}_\tau^{(p)}$ as $\tilde{c}_\tau^{(p)} \triangleq \frac{\sum_{q \in C_{p+1}^{[\Lambda]}} \dot{\boldsymbol{L}}_q \frac{p+1-|q\cap\tau|}{|\tau|+1-|q\cap\tau|}}{\sum_{\ell \in C_p^{[\Lambda]}} \dot{\boldsymbol{L}}_\ell}$.

Then, we define the subset of cardinality j that minimizes $\tilde{c}_\tau^{(p)}$ as $\tau_j^\star$, i.e., $\tau_j^\star \triangleq \operatorname{argmin}_{\tau\in 2^{[\Lambda]}:|\tau|=j} \tilde{c}_\tau^{(p)}$, and we define $\bar{a}_j$ as $\bar{a}_j \triangleq \sum_{\tau\in 2^{[\Lambda]}:|\tau|=j} a_\tau$. Then, it holds that

$$\sum_{\tau\in 2^{[\Lambda]}} c_\tau^{(p)} a_\tau \overset{(a)}{\geq} \sum_{\tau\in 2^{[\Lambda]}} \tilde{c}_\tau^{(p)} a_\tau \overset{(b)}{\geq} \sum_{j=0}^{\Lambda} \tilde{c}_{\tau_j^\star}^{(p)} \bar{a}_j, \tag{4.42}$$

where in (a) we have applied the fact that $\tilde{c}_\tau^{(p)} \leq c_\tau^{(p)}$ (cf. (4.37)), and in (b) we have used the definitions of $\tau_j^\star$ and $\bar{a}_j$.

We now provide a lemma that tells us the value of the optimal $\tau_j^\star$, for any $p \in [\Lambda]_0$ and $j \in [\Lambda]_0$.

Lemma 7. *Let us consider that the caches are sorted such that $L_1 \geq L_2 \geq \cdots \geq L_\Lambda$. Then, for any cardinality $|\tau| = j$, $j \in [\Lambda]$, it holds that*

$$\operatorname*{argmin}_{\substack{\tau\subseteq[\Lambda]\\|\tau|=j}} \tilde{c}_\tau^{(p)} \triangleq \begin{cases} \tau_j^\star = \{\emptyset\} & \text{if } j = 0 \\ \tau_j^\star = \{\Lambda - j + 1,\ \Lambda - j + 2,\ \ldots,\ \Lambda\} & \text{if } 1 \leq j < p \\ \tau_j^\star = \{1,\ 2,\ \ldots,\ j\} = [j] & \text{if } j \geq p \end{cases} \tag{4.43}$$

Note that for $j = p$, $\tilde{c}_\tau^{(p)}$ is the same for every τ such that $|\tau| = j$. This means that $\tilde{c}_{\tau_j^\star}^{(p)} = \tilde{c}_\tau^{(p)}\ \forall \tau : |\tau| = j$.

Proof. The proof is relegated to Appendix B.5. □

Now, jointly employing (4.42) in (4.40) and using the max-min inequality yields

$$\min_{\boldsymbol{Z}} \max_{p\in[\Lambda]_0} \sum_{\tau\in 2^{[\Lambda]}} c_\tau^{(p)} a_\tau \geq \max_{p\in[\Lambda]_0} \min_{\boldsymbol{Z}} \sum_{j=0}^{\Lambda} \tilde{c}_{\tau_j^\star}^{(p)} \bar{a}_j. \tag{4.44}$$

It then follows that

$$\begin{aligned} T^*(t, \mathbf{L}) \geq \max_{p\in\{0,1,\ldots,\Lambda\}} \min_{\bar{a}_j} & \sum_{j=0}^{\Lambda} \tilde{c}_{\tau_j^\star}^{(p)} \bar{a}_j \\ \text{subject to} \quad & \sum_{j=0}^{\Lambda} \bar{a}_j = 1 \\ & \sum_{j=0}^{\Lambda} j\bar{a}_j = t \\ & \bar{a}_j \geq 0, \quad \forall j \in [\Lambda]_0. \end{aligned} \tag{4.45}$$

We present now a result that will be instrumental for the following step in the derivation.

Proposition 2. *The sequence $\left\{\tilde{c}_{\tau_j^\star}^{(p)}\right\}$ is a decreasing sequence in $j \in [\Lambda]_0$.*

Proof. The proof is relegated to Appendix B.4. □

We know focus on the inner optimization problem in (4.45) for a fixed $p \in [\Lambda]_0$, and we follow the same steps as in [5] to solve it analytically. In this respect, we know from Proposition 2 that $\left\{\tilde{c}^{(p)}_{\tau^\star_j}\right\}$ is a decreasing sequence in $j \in [\Lambda]_0$, and thus its convex envelope is a decreasing and convex sequence. Thus, applying Jensen inequality, we obtain that

$$T^*(t, \mathbf{L}) \geq \max_{p \in [\Lambda]_0} \mathcal{T}^{(p,\mathbf{L})}_{low}(t) \tag{4.46}$$

where

$$\mathcal{T}^{(p,\mathbf{L})}_{low}(t) \triangleq \mathrm{Conv}_{j \in [\Lambda]_0}\left(\tilde{c}^{(p)}_{\tau^\star_j}\right). \tag{4.47}$$

Theorem 4 simply follows from the fact that

$$\max_{p \in \{0,1,\ldots,\Lambda\}} \mathcal{T}^{(p,\mathbf{L})}_{low}(t) \geq \mathcal{T}^{(\bar{t},\mathbf{L})}_{low}(t), \tag{4.48}$$

where $\bar{t} = \mathrm{round}(t)$.

Proof of case (i) in Theorem 5

We recall that, under the assumption of homogeneous caching, t is integer, thus implying that $\bar{t} = \mathrm{round}(t) = t$. For $p = t$, and under the aforementioned assumption, the problem in (4.45) reduces to

$$\begin{aligned} \min_{\bar{a}_j} \quad & \tilde{c}^{(t)}_{\tau^\star_t} \bar{a}_t \\ \text{subject to} \quad & \bar{a}_t = 1. \end{aligned} \tag{4.49}$$

It is easy to verify that, for all $\tau \in [\Lambda] : |\tau| = t$, it holds that

$$\tilde{c}^{(t)}_{\tau} = \tilde{c}^{(t)}_{\tau^\star_t} = \frac{\sum_{q \in C^{[\Lambda]}_{t+1}} \prod_{j=1}^{t+1} L_{q(j)}}{\sum_{q \in C^{[\Lambda]}_{t}} \prod_{j=1}^{t} L_{q(j)}}, \tag{4.50}$$

which, together with (4.49), directly results in

$$T^*(t, \mathbf{L}) \geq \tilde{c}^{(t)}_{\tau^\star_t}, \tag{4.51}$$

which concludes the proof of case (i) in Theorem 5. □

4.6 The Topology-Partially-Aware Scenario

In this section, we present the achievable scheme and the matching converse for the topology-partially-aware scenario previously described.

4.6.1 Achievable Scheme

With the knowledge of the cache occupancy vector $\bar{\mathbf{L}}$ at hand — which we recall again that can represent the expected cache occupancy vector in the delivery phase —, the server designs the memory allocation $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$ and cache placement $\boldsymbol{Z}_{\bar{\mathbf{L}}}^{*}$ as described in Section 4.4.1. For the subsequent delivery phase with topology described by $\mathbf{L}$, the following fact holds.

Proposition 3. *For any $(t+1)$-tuple $\mathcal{Q} \subset [\Lambda]$, the total number of subfiles with first subscript $\tau_\lambda = \mathcal{Q} \setminus \{\lambda\}$ that are missing from all users associated to any specific cache $\lambda \in \mathcal{Q}$ is equal to*

$$P_{\lambda_\mathcal{Q}} = L_\lambda \prod_{j=1}^{t} \bar{L}_{\tau_\lambda(j)}. \tag{4.52}$$

Proof. Considering cache λ, there are L_λ users requesting L_λ files. For each of these files there are $\prod_{j=1}^{t} \bar{L}_{\tau_\lambda(j)}$ subfiles with first index $\tau_\lambda = \mathcal{Q} \setminus \{\lambda\}$. □

In what follows, we will use $\mathbf{d}_\lambda$ to denote the vector of indices of the files requested by the users in $\mathcal{U}_\lambda$. Similarly to the delivery scheme in section 4.4.2, for a fixed $(t+1)$-tuple $\mathcal{Q}$ and any $\lambda \in \mathcal{Q}$, consider the set of subfiles, with first subscript $\tau = \mathcal{Q} \setminus \{\lambda\}$, that are requested from users in $\mathcal{U}_\lambda$, i.e.

$$\{W_{\tau,m}^{(\mathbf{d}_\lambda(j))} : j \in [L_\lambda], m \in A_\tau\},$$

where we recall that $A_\tau = \left\{1, 2, \ldots, \prod_{j=1}^{t} \bar{L}_{\tau(j)}\right\}$. From Proposition 3, we know that the cardinality of this set is $P_{\lambda_\mathcal{Q}}$, thus we can relabel these subfiles as

$$\mathcal{F}_{\lambda_\mathcal{Q}} = \{F_{\tau,j}^{(\lambda)} : j \in [P_{\lambda_\mathcal{Q}}]\}.$$

Let us now define the quantity $P_\mathcal{Q}^{(max)} \triangleq \max_{\lambda \in \mathcal{Q}} P_{\lambda_\mathcal{Q}}$ and $F_{\tau,j}^{(\lambda)} \triangleq \emptyset$ for any $\lambda \in \mathcal{Q}$, $\tau = \mathcal{Q} \setminus \{\lambda\}$ and $P_{\lambda_\mathcal{Q}} < j \leq P_\mathcal{Q}^{(max)}$. Because of the cache placement phase, we notice that for any $(t+1)$-tuple $\mathcal{Q}$ and any $j \in [P_\mathcal{Q}^{(max)}]$, the set of subfiles

$$F_{\mathcal{Q}\setminus\{\lambda\},j}^{(\lambda)} \ \forall \lambda \in \mathcal{Q} \tag{4.53}$$

forms a clique of $t+1$ nodes. For any $(t+1)$-tuple $\mathcal{Q} \in [\Lambda]$, we have $P_\mathcal{Q}^{(max)}$ cliques as in (4.53), all of $t+1$ nodes. Consequently, we transmit, for each $(t+1)$-tuple $\mathcal{Q} \subseteq [\Lambda]$, the following $P_\mathcal{Q}^{(max)}$ XORs

$$X_\mathcal{Q}(j) = \bigoplus_{\lambda \in \mathcal{Q}} F_{\mathcal{Q}\setminus\{\lambda\},j}^{(\lambda)}, \quad \forall j \in [P_\mathcal{Q}^{(max)}], \tag{4.54}$$

whose structure allows for clique-based decoding as in [2].

Delay Evaluation For each $(t+1)$-tuple $\mathcal{Q}$, the server transmits

$$P_{\mathcal{Q}}^{(max)} = \max_{\lambda \in \mathcal{Q}} P_{\lambda_{\mathcal{Q}}}$$

XORs. The total number of XORs sent through the channel is

$$\sum_{\mathcal{Q} \in C_{t+1}^{[\Lambda]}} \max_{\lambda \in \mathcal{Q}} P_{\lambda_{\mathcal{Q}}} = \sum_{\mathcal{Q} \in C_{t+1}^{[\Lambda]}} \max_{\lambda \in \mathcal{Q}} L_{\lambda} \prod_{j=1}^{t} \bar{L}_{\tau_{\lambda}(j)},$$

where $\tau_{\lambda} = \mathcal{Q} \setminus \{\lambda\}$. This result and the cache placement $\boldsymbol{Z}_{\bar{\mathbf{L}}}^{*}$ imply that the normalized delivery time of the achievable scheme for any $t, \mathbf{L}$ and $\bar{\mathbf{L}}$ is

$$T(t, \mathbf{L}, \bar{\mathbf{L}}) = \frac{\sum_{\mathcal{Q} \in C_{t+1}^{[\Lambda]}} \max_{\lambda \in \mathcal{Q}} L_{\lambda} \dot{\bar{\mathbf{L}}}_{\tau_{\lambda}}}{e_t(\bar{\mathbf{L}})}, \tag{4.55}$$

where we have applied the notation $\dot{\bar{\mathbf{L}}}_{\tau_{\lambda}} = \prod_{j=1}^{t} \bar{L}_{\tau_{\lambda}(j)}$, $e_t(\bar{\mathbf{L}}) = \sum_{\tau_{\lambda} \in C_t^{[\Lambda]}} \prod_{j=1}^{t} \bar{L}_{\tau_{\lambda}(j)}$.

4.6.2 Converse Bound

To develop the lower bound for $T^{*}(t, \mathbf{L}, \bar{\mathbf{L}})$, we immediately observe that, unlike in the topology-aware scenario, the cache placement is fixed in this case, thus resulting in a simpler problem. Before proceeding with the proof, we notice that, under the cache placement $\boldsymbol{Z}_{\bar{\mathbf{L}}}^{*}$, for any $n \in [N]$, we can put together all subfiles stored in the caches in set $\tau \subset [\Lambda] : |\tau| = t$ and form the subfile $W_{\tau}^{(n)} = \{W_{\tau, m_{\tau}}^{(n)} : m_{\tau} \in A_{\tau}\}$.

Under the cache placement $\boldsymbol{Z}_{\bar{\mathbf{L}}}^{*}$, Lemma 5 also holds for the considered topology-partially-aware scenario, such that $T^{*}(\boldsymbol{Z}_{\bar{\mathbf{L}}}^{*}, \mathbf{d}, \mathbf{L})$ can be lower bounded as

$$T^{*}(\boldsymbol{Z}_{\bar{\mathbf{L}}}^{*}, \mathbf{d}, \mathbf{L}) \geq \sum_{\lambda=1}^{\Lambda} \sum_{\ell=1}^{L_{\sigma(\lambda)}} \sum_{\tau_{\lambda} \subseteq [\Lambda] \setminus \{\sigma(1), \ldots, \sigma(\lambda)\}} |W_{\tau_{\lambda}}^{(\mathbf{d}_{\sigma(\lambda)}(\ell))}|, \tag{4.56}$$

$$= \sum_{\lambda=1}^{\Lambda} L_{\sigma(\lambda)} \sum_{q \in C_t^{[\Lambda] \setminus \{\sigma(1), \ldots, \sigma(\lambda)\}}} \frac{\dot{\bar{\mathbf{L}}}_q}{e_t(\bar{\mathbf{L}})}, \tag{4.57}$$

where (4.57) follows directly from the fact that, under the cache placement $\boldsymbol{Z}_{\bar{\mathbf{L}}}^{*}$, we have that

$$|W_{\tau}^{(n)}| = \begin{cases} \frac{\dot{\bar{\mathbf{L}}}_{\tau}}{e_t(\bar{\mathbf{L}})} & |\tau| = t \\ 0 & \text{otherwise} \end{cases} \qquad \forall n \in [N]$$

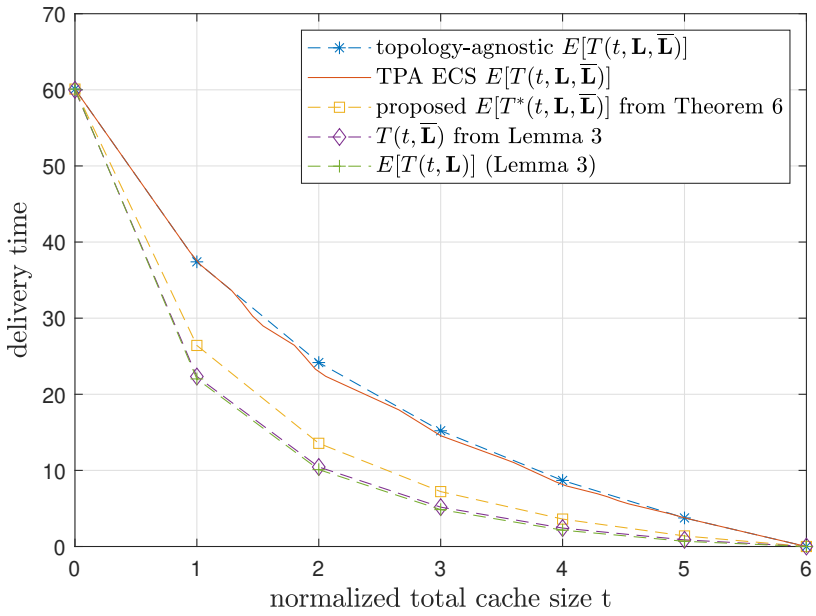


Figure 4.2 – Delivery time comparisons for $\mathbf{L} = (20, 20, 8, 6, 4, 2)$

Now, we first note that (4.57) does not depend on the specific demand $\mathbf{d}$. Then, we maximize over all possible caches permutations σ to obtain

$$T^*(t,\mathbf{L},\bar{\mathbf{L}}) = \max_{\mathbf{d}} T^*(\boldsymbol{Z}^*_{\bar{\mathbf{L}}},\mathbf{d},\mathbf{L}) \tag{4.58}$$

$$\geq \max_{\sigma\in S_\Lambda} \sum_{\lambda=1}^{\Lambda} L_{\sigma(\lambda)} \sum_{q\in C_t^{[\Lambda]\setminus\{\sigma(1),\ldots,\sigma(\lambda)\}}} \frac{\dot{\bar{\mathbf{L}}}_q}{e_t(\bar{\mathbf{L}})} \tag{4.59}$$

$$= \max_{\sigma\in S_{\Lambda,\Lambda-t}} \sum_{\lambda=1}^{\Lambda-t} L_{\sigma(\lambda)} \sum_{q\in C_t^{[\Lambda]\setminus\{\sigma(1),\ldots,\sigma(\lambda)\}}} \frac{\dot{\bar{\mathbf{L}}}_q}{e_t(\bar{\mathbf{L}})} \tag{4.60}$$

where we recall that $S_{\Lambda,\Lambda-t}$ is the set of $(\Lambda-t)$-permutations of $[\Lambda]$.

4.7 Numerical Results

We here focus on the topology-partially-aware scenario, and we assume in the delivery phase that the cache occupancy vector $\mathbf{L}$ is the realization of a collection of independent random variables $\boldsymbol{\mathcal{L}} = (\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_\Lambda)$ with expected value $\bar{\mathbf{L}} = (\bar{L}_1, \bar{L}_2, \ldots, \bar{L}_\Lambda)$, where we have used $\bar{L}_\lambda = \mathbb{E}[\mathcal{L}_\lambda], \lambda \in [\Lambda]$.

Fact 1. For the ($\boldsymbol{\mathcal{L}}$,t) topology-partially-aware scenario, the expected delivery time satisfies

$$\mathbb{E}[T^*(t,\mathbf{L},\bar{\mathbf{L}})] \geq T(t,\bar{\mathbf{L}}). \tag{4.61}$$

Proof.

$$\mathbb{E}[T^*(t,\mathbf{L},\bar{\mathbf{L}})] = \mathbb{E}\left[\max_{\sigma\in S_{\Lambda,\Lambda-t}} \sum_{\lambda=1}^{\Lambda-t} L_{\sigma(\lambda)} \sum_{q\in C_t^{[\Lambda]}\setminus\{\sigma(1),\ldots,\sigma(\lambda)\}} \frac{\dot{\bar{L}}_q}{e_t(\bar{\mathbf{L}})}\right]$$

$$\geq \max_{\sigma\in S_{\Lambda,\Lambda-t}} \mathbb{E}\left[\sum_{\lambda=1}^{\Lambda-t} L_{\sigma(\lambda)} \sum_{q\in C_t^{[\Lambda]}\setminus\{\sigma(1),\ldots,\sigma(\lambda)\}} \frac{\dot{\bar{L}}_q}{e_t(\bar{\mathbf{L}})}\right] \tag{4.62}$$

$$= \max_{\sigma\in S_{\Lambda,\Lambda-t}} \sum_{\lambda=1}^{\Lambda-t} \mathbb{E}\left[L_{\sigma(\lambda)}\right] \sum_{q\in C_t^{[\Lambda]}\setminus\{\sigma(1),\ldots,\sigma(\lambda)\}} \frac{\dot{\bar{L}}_q}{e_t(\bar{\mathbf{L}})} \tag{4.63}$$

$$= \frac{\sum_{\mathcal{Q}\in C_{t+1}^{[\Lambda]}} \max_{\lambda\in\mathcal{Q}} \mathbb{E}\left[L_\lambda\right] \dot{\bar{\mathbf{L}}}_{\mathcal{Q}\setminus\{\lambda\}}}{e_t(\bar{\mathbf{L}})} \tag{4.64}$$

$$= \frac{\sum_{\mathcal{Q}\in C_{t+1}^{[\Lambda]}} \max_{\lambda\in\mathcal{Q}} \bar{L}_\lambda \dot{\bar{\mathbf{L}}}_{\mathcal{Q}\setminus\{\lambda\}}}{e_t(\bar{\mathbf{L}})}$$

$$= \frac{e_{t+1}(\bar{\mathbf{L}})}{e_t(\bar{\mathbf{L}})}$$

$$= T(t,\bar{\mathbf{L}}), \tag{4.65}$$

where (4.63) follows from the independence of the random variables $\{\mathcal{L}_\lambda\}$ and (4.64) is proven in Appendix B.6. □

For the above setting, we will consider 3 different schemes. The first is the single-antenna topology-agnostic scheme of Section 3.3.1 (equivalent to the one in [32]), which does not exploit the knowledge of $\bar{\mathbf{L}}$ for the cache placement, which is done according to MAN cache placement. The second scheme, which we will refer to as *TPA ECS scheme* (topology-partially-aware equal-cache-size scheme), is the one achieving the delivery time in equation (24a) of [59] for the case when there are no cache-less users. We notice that [59] assumes that all the caches have the same size, which cannot be optimized. The cache placement of the aforementioned scheme in [59] partitions the set of caches in G groups such that all the caches in the same group store the same content, and it applies MAN placement for G caches/users. If the cache occupancy vector $\mathbf{L}$ is known in advance during placement, the best partition is chosen by leveraging $\mathbf{L}$ in order to minimize the delivery time. In the topology-partially-aware scenario considered here, $\mathbf{L}$ cannot be known in advance, thus we select the partition according to the average cache occupancy vector $\bar{\mathbf{L}}$. Delivery is performed by means of the multi-round scheme for the single-antenna setting in Section 3.3.1 (se also [32]). Finally, the third scheme is our proposed topology-partially-aware scheme achieving the delivery time in Theorem 6. We assume that each random variable $\mathcal{L}_\lambda$ follows a poisson distribution with mean $\bar{L}_\lambda$, i.e. $\mathcal{L}_\lambda \sim \text{Poiss}(\bar{L}_\lambda)$, and we consider the scenario where the expected number of users per cache is $\bar{\mathbf{L}} = (20, 20, 8, 6, 4, 2)$. Figure 4.2 shows the average delivery time achieved by the three schemes, the memory-rate curve $(t, T(t,\bar{\mathbf{L}}))$ from Lemma 3, and the average

memory-rate curve $(t, \mathbb{E}[T(t, \mathbf{L})])$. It is evident that the proposed scheme with optimized shared caches largely outperforms the other two scheme, thus highlighting the importance of proper memory allocation. The plot also confirms Fact 1, and it shows that $T(t, \bar{\mathbf{L}})$ is a good approximation of $\mathbb{E}[T(t, \mathbf{L})]$, i.e. $T(t, \bar{\mathbf{L}}) \approx \mathbb{E}[T(t, \mathbf{L})]$.

Chapter 5

Multi-Access Shared-Cache Networks

This chapter considers the K-user cache-aided shared-link channel where each user has access to exactly z neighbouring caches of normalized size γ, and where each cache assists exactly z users. For this setting, for two opposing memory regimes, we propose novel caching and coded delivery schemes which maximize the local caching gain, and achieve a *coding gain* larger than $K\gamma + 1$ despite the fact that the cache replication factor remains $K\gamma$ irrespective of z. Interestingly, when $z = \frac{K-1}{K\gamma}$, the derived optimal coding gain is $K\gamma z + 1$, matching the performance of a hypothetical scenario where each user has its own dedicated cache of size $z\gamma$.

Previous Works on Multi-Access Coded Caching

The[1] shared-cache setting where each user has access to more than one cache was explored for the first time in [26] which considered a K-user shared-link setting, where each user is assisted by exactly $z > 1$ caches, and where each cache can serve exactly z users. In this context, the work in [26] provided a caching and delivery strategy whose centralized variant[2] achieves a worst-case NDT of

$$T = \frac{K(1 - z\gamma)}{K\gamma + 1} \tag{5.1}$$

reflecting an ability to increase the local caching gain to $z\gamma$ (meaning that each user sees a fraction $z\gamma$ of each file), as well as an ability to preserve the coding gain to $K\gamma + 1$, by virtue of a scheme that served $K\gamma + 1$ users at a time. For this same setting, in [66] the authors provide explicit designs for $K = 4$, $N = 4$, $z = \{2, 3\}$, $M = 1$ and $K = 6$, $N = 6$, $z = 3$, $M = 1$, for which matching lower bounds are also developed to prove optimality of the schemes under uncoded cache placement. Finally, the authors provide a scheme for the extreme case of $z = K - 1$.

[1] In this paragraph we mentioning only the works that had been published before the submission of our results to the IEEE Information Theory Workshop 2019. After our contributions were published, a significant set of results on multi-access coded caching appeared. We briefly mention these papers in the concluding section of this chapter.

[2] The work in [26] proposed a decentralized cache placement scheme, whose performance is slightly reduced over the easy-to-extend-to centralized variant whose delivery time we recorded above.

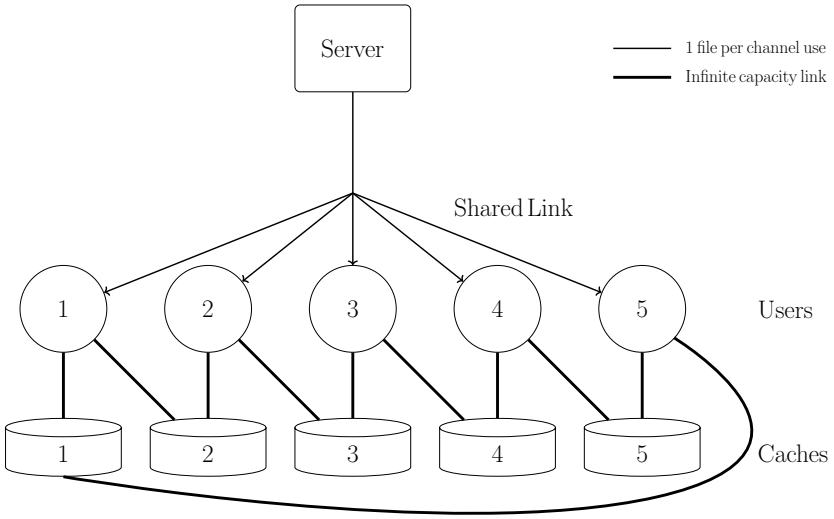


Figure 5.1 – Multi-Access setting for $K = 5$ and $z = 2$.

5.1 System Model and Problem Definition

We consider a network where K users are connected via an error-free shared link to a server storing N ($N \geq K$) files $W^{(1)}, W^{(2)}, \ldots, W^{(N)}$. Each user has access to z out of K helper caches[3], each of size $M = N\gamma$ (units of file), where $\gamma \in \{\frac{1}{K}, \frac{2}{K}, \ldots, 1\}$. We will use $\mathcal{Z}_k$ to denote the content in cache k, and we will assume that each user has an unlimited capacity link to the caches it is connected to. Reflecting the assumption that each user has access to exactly z caches and each cache connects to exactly z users, we will consider the symmetric topology where each user $k \in [K]$ is associated to caches

$$\mathcal{C}_k \triangleq \langle k, k+1, \ldots, k+z-1 \rangle \subseteq [K]$$

where in the above we use the notation $\langle \mathcal{M} \rangle = \{m \mid m \text{ for } m \leq K \ ; \ m-K \text{ for } m > K, \ m \in \mathbb{Z}^+, \ \forall m \in \mathcal{M}\}$. A pictorial representation of the studied setting can be found in Figure 5.1.

The system works in two phases: the cache placement phase and the delivery phase. The first consists of filling the caches with the content of the library without knowledge of the users' demands. In the delivery phase, each user k requests a file from the library. We denote the index of such file by d_k and we collect all the indices of the requested files in the demand vector $\mathbf{d} = (d_1, d_2, \ldots, d_K)$. The work focuses on the worst case where all users request different files. Upon reception of the demand vector $\mathbf{d}$, the server will transmit a message X that spans T units of time. Each user k will use X and their own available cache content $\cup_{i \in \mathcal{C}_k} \mathcal{Z}_i$ to decode the desired file $W^{(d_k)}$. Our objective is to provide a caching and delivery scheme that reduces the delivery delay T.

[3]Notice that the assumption of having as many users as caches can be relaxed to a more general case where the system has more users than caches, with a potential non uniform distribution of the users among the caches. Under these circumstances, the schemes described here can be combined with the topology-agnostic scheme (for the single antenna setting) presented in Chapter 3.

$$X_1 = \sum_{k=1}^{K} \left[\sum_{j=1}^{\min\{k-z-1,z-1\}} \left[\left\lfloor \frac{K-(k+z+j)}{2} \right\rfloor + 1 \right]_+ + \sum_{j=\max\{1,k-z\}}^{z-1} \left[\left\lfloor \frac{K-2z-2j}{2} \right\rfloor + 1 \right]_+ \right],$$

$$X_2 = K \left[\frac{\left\lfloor \frac{K-2-z}{3} \right\rfloor - z}{2} + 1 \right]_+ \left(K-4z+3-3 \left[\frac{\left\lfloor \frac{K-2-z}{3} \right\rfloor - z}{2} + 1 \right]_+ \right), \quad S = \frac{K(K-2z+2)}{4}. \tag{5.2}$$

5.2 Main Results

This section presents the main results. The proof of the following theorem follows from the scheme described in Section 5.3.

Theorem 7. *In the coded caching setting where each user is connected to z consecutive caches, when $K\gamma = 2$, the delivery time satisfies*

$$\frac{K-2z}{4} < T(2,z) \leq \frac{K-2z}{3}, \tag{5.3}$$

and its exact value is $T(2,z) = \frac{X_1+X_2}{S}$, where[4] X_1, X_2 and S are given in (5.2).

Remark 7. *The scheme consists of transmissions serving either $K\gamma + 1 = 3$ users or 4 users at a time. Figure 5.2 shows, for several values of z, the achievable DoF as a function of K. We can see how the DoF always exceeds $K\gamma + 1 = 3$ and can approach 4 for some values of K and z.*

We now characterize the optimal worst-case delivery time (under the assumption of uncoded cache placement) for the case of $z = \frac{K-1}{K\gamma}$, for any $K\gamma$.

Theorem 8. *In the addressed K-user caching network with access to an integer number $z = \frac{K-1}{K\gamma}$ of caches of normalized size γ, the optimal delivery time, under the assumption of uncoded cache placement, takes the form*

$$T^* = \frac{K(1-\gamma z)}{K\gamma z + 1} = \frac{1}{K} \tag{5.4}$$

corresponding to a DoF of $K\gamma z + 1$ users served at a time.

The scheme that achieves the above performance is described in Section 5.4. The optimality — under uncoded cache placement — follows directly from the fact that the achieved performance matches the optimal performance (cf. [52], [5]) of a dedicated-cache coded caching setting (identical to that in [2]), where each cache has an augmented size equal to $z\gamma$.

[4]The above expression holds for the case where $S(K-2z) - 4X_1$ is non negative and divisible by 3.

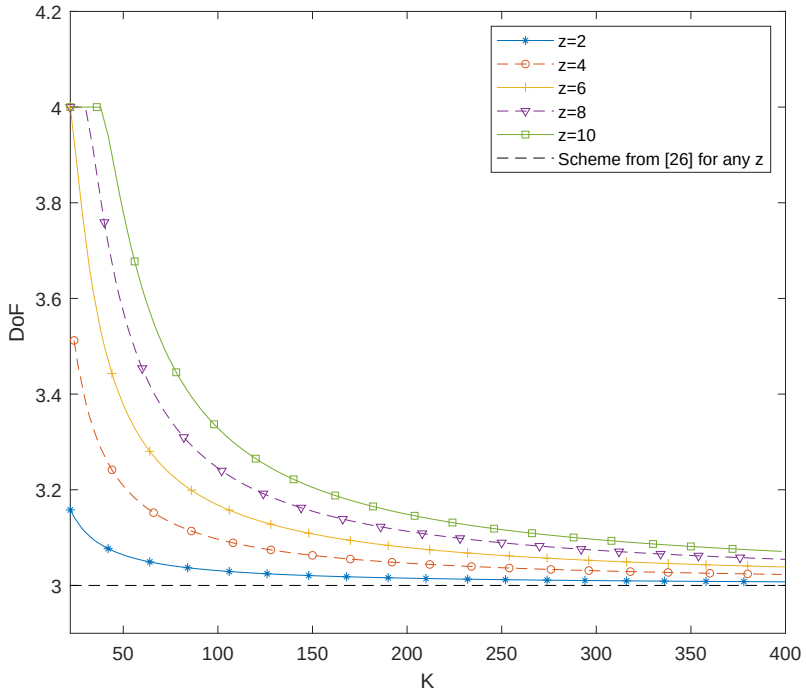


Figure 5.2 – Achievable sum DoF as a function of K, for several z. Case of $K\gamma = 2$.

5.3 Achievable Scheme for $K\gamma = 2$

In this section we present a caching and delivery scheme for the case of $K\gamma = 2$. The scheme preserves the full local caching gain as in [26], and achieves a coding gain that strictly exceeds $K\gamma + 1$.

5.3.1 Cache Placement Scheme

In the cache placement phase, we first split each file $W^{(n)}$ into $S = \frac{K(K-2z+2)}{4}$ subfiles $W^{(n)}_{\mathcal{T}}$ for each pair $\mathcal{T} \triangleq \{\mathcal{T}_1, \mathcal{T}_2\}$ from the set

$$\Psi \triangleq \{\mathcal{T} : \mathcal{T}_1 \in [K-z], \mathcal{T}_2 \in [\mathcal{T}_1 + z : 2 : \min\{K - z + \mathcal{T}_1, K\}]\} \tag{5.5}$$

where in the above we used the notation $[a : b : c]$ to denote an ordered set of integers[5], from a to c, in additive steps of b. After splitting the files, each cache k is filled as follows

$$\mathcal{Z}_k = \{W^{(n)}_{\mathcal{T}} \mid \forall n \in [N], \forall \mathcal{T} \ni k\}. \tag{5.6}$$

Verifying the memory constraint To show that the cache placement satisfies the per-cache memory $M = \frac{2N}{K}$, we focus without loss of generality on cache 1, and note that the

[5]Note that b may be negative.

number of subfiles (per file) in this cache is $\frac{K-2z}{2}+1$. Recalling that each such subfile is of size $1/S$, yields

$$\frac{N\left(\frac{K-2z}{2}+1\right)}{\frac{K(K-2z+2)}{4}} = \frac{2N}{K} = M$$

thus proving that the memory constraint is satisfied.

5.3.2 Delivery Scheme

The delivery scheme has two phases, where the first phase transmits XORs composed of 4 subfiles, while the second phase transmits XORs composed of $K\gamma + 1 = 3$ subfiles.

Phase 1

Recall from above that any subfile $W_{\mathcal{T}}^{(d_k)},\ k \in \mathcal{T} \in \Psi$ ($\mathcal{T}_1 \in [k:1:\langle k+z-1\rangle]$), is already available at one of the caches seen by the requesting user $k \in [K]$. For each user $k \in [K]$, the aim of this first phase is to serve the subfiles in the set

$$\left\{W_{\mathcal{T}_1,\mathcal{T}_2}^{(d_k)} \mid \forall\{\mathcal{T}_1,\mathcal{T}_2\} \in \Psi : \mathcal{T}_1 \in \langle[k-z+1{:}1{:}k-1]\rangle \cup \langle[k+z{:}1{:}k+2z-2]\rangle\right\}. \tag{5.7}$$

For any $k \in [K]$, let us define the following two sets

$$\begin{aligned}\Omega_{k,1} &\triangleq [k+1:1:k+z-1]\\ \Omega_{k,2} &\triangleq [k-z+1:1:k-1]\end{aligned} \tag{5.8}$$

and the set

$$B_{k,j} \triangleq [\Omega_{k,1}(j)+z:2:U_{k,j}] \tag{5.9}$$

where

$$U_{k,j} \triangleq \begin{cases} \langle\Omega_{k,2}(j)-z\rangle, & \text{if } \Omega_{k,2}(j)-z<0\\ K, & \text{otherwise.}\end{cases} \tag{5.10}$$

and where in the above we used the notation $\Gamma(j)$ to denote the j-th element of an ordered set Γ. Next, for any $k \in [K]$ and any $j \in [z-1]$ we form the following XOR

$$\begin{aligned}X(k,j,m) = W_{\langle\Omega_{k,1}(j)\rangle,B_{k,j}(m)}^{(d_{\langle k-z+1\rangle})} \oplus W_{\langle\Omega_{k,2}(j)\rangle,B_{k,j}(m)}^{(d_k)}\\ \oplus W_{\langle\Omega_{B_{k,j}(m),1}(j)\rangle,k}^{(d_{\langle B_{k,j}(m)-z+1\rangle})} \oplus W_{\langle\Omega_{B_{j,k}(m),2}(j)\rangle,k}^{(d_{B_{k,j}(m)})}.\end{aligned} \tag{5.11}$$

Creating the above XORs for every $m \in [|B_{k,j}|]$ and every $k \in [K]$, spans the entire set of requested files in (5.7), and what we show below is that each component subfile (in the XORs) can be successfully decoded by its corresponding user.

Decoding Consider any XOR as in (5.11) and let us focus on the subfiles $W^{(d_{\langle k-z+1\rangle})}_{\Omega_{k,1}(j),B_{k,j}(m)}$ and $W^{(d_k)}_{\Omega_{k,2}(j),B_{k,j}(m)}$ which are desired by users $\langle k-z+1\rangle$ and k, respectively. By the cache placement phase, we notice that the subfiles $W^{(d_{\langle B_{k,j}(m)-z+1\rangle})}_{\mathcal{T}_{B_{k,j}(m),z,1}(j),k}$ and $W^{(d_{B_{k,j}(m)})}_{\mathcal{T}_{b_{j,k,z}(m),z,2}(j),k}$ are both cached in cache k, thus enabling both users $\langle k-z+1\rangle$ and k to subtract these subfiles from $X(k,j,m)$. Next, we also notice that user $\langle k-z+1\rangle$ can cache out $W^{(d_k)}_{\Omega_{k,2}(j),B_{k,j}(m)}$ since $\Omega_{k,2}(j) \in \cup_{i\in\mathcal{C}_{\langle k-z+1\rangle}}\mathcal{Z}_i$. Similarly, user k can remove $W^{(d_{\langle k-z+1\rangle})}_{\Omega_{k,1}(j),B_{k,j}(m)}$ from $X(k,j,m)$ because $\Omega_{k,1}(j) \in \cup_{i\in\mathcal{C}_k}\mathcal{Z}_i$. Hence, we conclude that any XOR in (5.11) is decodable by both users k and $\langle k-z+1\rangle$. In the same way, it can be shown that users $\langle B_{k,j}(m)-z+1\rangle$ and $B_{k,j}(m)$ can successfully decode their own requested subfiles.

Phase 2

We start by defining the set

$$\delta = \left[z : 2 : \left\lfloor \frac{K-2-z}{3} \right\rfloor\right]$$

as well as the following set of triplets

$$\Theta = \Bigg\{(\theta_1,\theta_2,\theta_3)|\theta_1 = \delta(j), \theta_2 = 2\delta(j)+z+2(i-1),$$
$$\theta_3 = \delta(j)+2(i-1), \quad j\in[|\delta|],\ i \in \left[\frac{K-3\delta(j)-z}{2}\right]\Bigg\}. \tag{5.12}$$

For each triplet $\theta\in\Theta$ and for each $p\in[K]$, we generate the following two XORs

$$Y_p(\theta,1) = W^{(d_p)}_{\langle\theta_2-\theta_1+p-1\rangle,\langle\theta_2+p-1\rangle} \oplus W^{(d_{\langle\theta_2-\theta_1-z+p\rangle})}_{p,\langle\theta_2+p\rangle} \oplus W^{(d_{\langle\theta_2+p-1\rangle})}_{\langle z-2+p\rangle,\langle z-2+\theta_3+p\rangle} \tag{5.13}$$

$$Y_p(\theta,2) = W^{(d_p)}_{\langle\theta_2-\theta_1+p\rangle,\langle\theta_2+p\rangle} \oplus W^{(d_{\langle\theta_2-\theta_1-z+1+p\rangle})}_{p,\langle\theta_2+p\rangle} \oplus W^{(d_{\langle\theta_2+p-1\rangle})}_{\langle z+p-1\rangle,\langle z+\theta_3+p-1\rangle}. \tag{5.14}$$

As we did for phase 1, below we show that each subfile in the above XORs from (5.13) and (5.14) can be decoded successfully by their requesting user.

Decoding For any $p\in[K]$, we will prove that the subfiles in $Y_p(\theta,1)$ can be decoded by their intended users. The decodability proof for $Y_p(\theta,2)$ will then follow directly.

User p can cache out subfiles $W^{(d_{\langle\theta_2-\theta_1-z+p\rangle})}_{p,\langle p+\theta_2\rangle}$ and $W^{(d_{\langle\theta_2+p-1\rangle})}_{\langle z-2+p\rangle,\langle z-2+\theta_3+p\rangle}$ from $Y_p(\theta,1)$ since their subscripts $p\in\cup_{i\in\mathcal{C}_p}\mathcal{Z}_i$ and $\langle z-2+p\rangle\in\cup_{i\in\mathcal{C}_p}\mathcal{Z}_i$ correspond to the caches

that user p is connected to. Next, we notice that user $\langle \theta_2 - \theta_1 - z + p \rangle$ is connected to cache $\langle \theta_2 - \theta_1 + p - 1 \rangle$ and thus it can cache out subfile $W^{(d_p)}_{\langle \theta_2 - \theta_1 + p - 1 \rangle, \langle \theta_2 + p - 1 \rangle}$. The same user $\langle \theta_2 - \theta_1 - z + p \rangle = \langle \delta(j) + p + 2(i-1) \rangle$ has access to subfiles with subscripts in the set $\langle [\delta(j) + p + 2(i-1) : 1 : \delta(j) + p + 2(i-1) + z - 1] \rangle$. A relabelling of $\langle \theta_3 + p + z - 2 \rangle$ to $\langle \delta(j) + z + p - 2 + 2(i-1) \rangle$ highlights that user $\langle \theta_2 - \theta_1 - z + p \rangle$ can also remove $W^{(d_{\langle \theta_2 + p - 1 \rangle})}_{\langle z - 2 + p \rangle, \langle z - 2 + \theta_3 + p \rangle}$ from $Y_p(\theta, 1)$, and hence obtains its desired subfile $W^{(d_{\langle \theta_2 - \theta_1 - z + p \rangle})}_{p, \langle \theta_2 + p \rangle}$ successfully. Finally, we recall that user $\langle \theta_2 + p - 1 \rangle$ has access to caches $\mathcal{C}_{\langle \theta_2 + p - 1 \rangle} = \langle [\theta_2 + p - 1 : 1 : \theta_2 + p - 2 + z] \rangle$ and hence can successfully decode its desired subfile since it can cache out subfiles $W^{(d_p)}_{\langle \theta_2 - \theta_1 + p - 1 \rangle, \langle \theta_2 + p - 1 \rangle}$ and $W^{(d_{\langle \theta_2 - \theta_1 - z + p \rangle})}_{p, \langle \theta_2 + p \rangle}$.

Performance of the algorithm

We observe that each generated XOR serves a different set of 3 or 4 subfiles, which can all be decoded. Therefore, we can conclude that the achieved sum DoF is between 3 and 4. From the description of the algorithm, it can be seen that the proposed delivery scheme is valid for any demand vector $\mathbf{d}$. Following the construction, we can readily count the total number of XORs transmitted during phase 1 and phase 2 to respectively be X_1 and X_2 from (5.2), which concludes the proof of the achievable delay in Theorem 7.

5.3.3 Illustrative Example

In this subsection we offer an example that may help to better understand the scheme. We consider the setting with parameters $K = 10$, $K\gamma = 2$, and $z = 2$. For the sake of simplicity, we will use 0 to represent the index 10 and also, when describing a double index i, j, we will omit the comma.

In the placement phase, we first split each file, according to (5.5), into $S = 20$ equally-sized subfiles with indices

$$\begin{aligned}\Psi = &\{13, 15, 17, 19, 24, 26, 28, 20, 35, 37, 39, 46, 48, 40, 57,\\ &59, 68, 60, 79, 80\}\end{aligned}$$

and we then fill the caches according to (5.6) as follows

$$\begin{aligned}\mathcal{Z}_1 &= \{W^{(n)}_{13}, W^{(n)}_{15}, W^{(n)}_{17}, W^{(n)}_{19},\ \forall n \in [N]\}\\ \mathcal{Z}_2 &= \{W^{(n)}_{24}, W^{(n)}_{26}, W^{(n)}_{28}, W^{(n)}_{20},\ \forall n \in [N]\}\\ &\vdots\\ \mathcal{Z}_9 &= \{W^{(n)}_{19}, W^{(n)}_{39}, W^{(n)}_{59}, W^{(n)}_{79},\ \forall n \in [N]\}\\ \mathcal{Z}_0 &= \{W^{(n)}_{20}, W^{(n)}_{40}, W^{(n)}_{60}, W^{(n)}_{80},\ \forall n \in [N]\}.\end{aligned}$$

We notice that the cache placement guarantees an empty intersection $\mathcal{Z}_k \cap \mathcal{Z}_{\langle k+1 \rangle} = \emptyset$ of any $z = 2$ neighboring caches, and thus a full local caching gain ($z\gamma = 0.4$).

In the delivery phase we consider the worst-case demand vector $\mathbf{d} = (1, 2, \ldots, 9, 0)$. We will list the XORs of phase 1 and phase 2, but before doing that, let us offer some intuition on the design of the XORs of the first phase.

Let us consider a pair of users (say, users 0 and 1) that "see" a common cache (in this case, cache 1). For these two users we will create a generic XOR

$$W^{(0)}_{\sigma_1,\sigma_2} \oplus W^{(1)}_{\tilde{\sigma}_1,\tilde{\sigma}_2} \tag{5.15}$$

which will be combined with another XOR

$$W^{(3)}_{\tau_1,\tau_2} \oplus W^{(4)}_{\tilde{\tau}_1,\tilde{\tau}_2} \tag{5.16}$$

which is meant for another pair of users, say 3 and 4, that again share a common cache (cache 4). Combining the two XORs yields a new XOR $X = W^{(0)}_{\sigma_1,\sigma_2} \oplus W^{(1)}_{\tilde{\sigma}_1,\tilde{\sigma}_2} \oplus W^{(3)}_{\tau_1,\tau_2} \oplus W^{(4)}_{\tilde{\tau}_1,\tilde{\tau}_2}$ of 4 subfiles. To guarantee decoding for all, we will set $\sigma_1 = \tilde{\sigma}_1 = 4$ to let user 3 and user 4 "cache out" from X the subfiles in (5.15) and similarly we set $\tau_1 = \tilde{\tau}_1 = 1$ in order to let users 0 and 1 cache out the XOR in (5.16). Next, we choose $\sigma_2 = 2$ so that user 1 can remove subfile $W^{(0)}_{4,2}$ from X and $\tilde{\sigma}_2 = 0$ to let user 0 remove subfile $W^{(1)}_{4,0}$ from X. A similar choice of τ_2 and $\tilde{\tau}_2$ will result in[6]

$$X = W^{(0)}_{24} \oplus W^{(1)}_{40} \oplus W^{(3)}_{15} \oplus W^{(4)}_{13}. \tag{5.17}$$

The list of XORs sent during phase 1 is given below.

$$\begin{aligned}
X(1,1,1) &= W^{(0)}_{24} \oplus W^{(1)}_{40} \oplus W^{(3)}_{15} \oplus W^{(4)}_{13} \\
X(1,1,2) &= W^{(0)}_{26} \oplus W^{(1)}_{60} \oplus W^{(5)}_{17} \oplus W^{(6)}_{15} \\
X(1,1,3) &= W^{(0)}_{28} \oplus W^{(1)}_{80} \oplus W^{(7)}_{19} \oplus W^{(8)}_{17} \\
X(2,1,1) &= W^{(1)}_{35} \oplus W^{(2)}_{15} \oplus W^{(4)}_{26} \oplus W^{(5)}_{24} \\
X(2,1,2) &= W^{(1)}_{37} \oplus W^{(2)}_{17} \oplus W^{(6)}_{28} \oplus W^{(7)}_{26} \\
X(2,1,3) &= W^{(1)}_{39} \oplus W^{(2)}_{19} \oplus W^{(8)}_{20} \oplus W^{(9)}_{28} \\
X(3,1,1) &= W^{(2)}_{46} \oplus W^{(3)}_{26} \oplus W^{(5)}_{37} \oplus W^{(6)}_{35} \\
X(3,1,2) &= W^{(2)}_{48} \oplus W^{(3)}_{28} \oplus W^{(7)}_{39} \oplus W^{(8)}_{37} \\
X(3,1,3) &= W^{(2)}_{40} \oplus W^{(3)}_{20} \oplus W^{(9)}_{13} \oplus W^{(0)}_{39} \\
X(4,1,1) &= W^{(3)}_{57} \oplus W^{(4)}_{37} \oplus W^{(6)}_{48} \oplus W^{(7)}_{46} \\
X(4,1,2) &= W^{(3)}_{59} \oplus W^{(4)}_{39} \oplus W^{(8)}_{40} \oplus W^{(9)}_{48} \\
X(5,1,1) &= W^{(4)}_{68} \oplus W^{(5)}_{48} \oplus W^{(7)}_{59} \oplus W^{(8)}_{57} \\
X(5,1,2) &= W^{(4)}_{60} \oplus W^{(5)}_{40} \oplus W^{(9)}_{15} \oplus W^{(0)}_{59}
\end{aligned}$$

[6]This intuition can be generalized for $z > 2$ and used as a baseline in the general description of the scheme presented in Section 5.3.2.

$$X(6,1,1) = W_{79}^{(5)} \oplus W_{59}^{(6)} \oplus W_{60}^{(8)} \oplus W_{68}^{(9)}$$
$$X(7,1,1) = W_{80}^{(6)} \oplus W_{60}^{(7)} \oplus W_{17}^{(9)} \oplus W_{79}^{(0)}.$$

In phase 2, the $X_2 = 20$ transmissions from (5.13) and (5.14) are cyclically generated as shown below.

$$\begin{aligned}
Y_1(\theta,1) &= W_{46}^{(1)} \oplus W_{17}^{(3)} \oplus W_{13}^{(6)},\\
Y_2(\theta,1) &= W_{57}^{(2)} \oplus W_{28}^{(4)} \oplus W_{24}^{(7)}\\
Y_3(\theta,1) &= W_{68}^{(3)} \oplus W_{39}^{(5)} \oplus W_{35}^{(8)}\\
&\vdots\\
Y_0(\theta,1) &= W_{35}^{(0)} \oplus W_{60}^{(2)} \oplus W_{20}^{(5)}
\end{aligned}$$

and

$$\begin{aligned}
Y_1(\theta,2) &= W_{57}^{(1)} \oplus W_{17}^{(4)} \oplus W_{24}^{(6)}\\
Y_2(\theta,2) &= W_{68}^{(2)} \oplus W_{28}^{(5)} \oplus W_{35}^{(7)}\\
Y_3(\theta,2) &= W_{79}^{(3)} \oplus W_{39}^{(6)} \oplus W_{46}^{(8)}\\
&\vdots\\
Y_0(\theta,2) &= W_{46}^{(0)} \oplus W_{60}^{(3)} \oplus W_{13}^{(5)}.
\end{aligned}$$

In the end, we have $S = 20$, $X_1 = 15$ and $X_2 = 20$ which gives

$$T = \frac{X_1 + X_2}{S} = \frac{35}{20}$$

and a coding gain $K(1 - z\gamma)/T = 3.43$.

5.4 Achievable Scheme for $K = K\gamma z + 1$

Corresponding to Theorem 8, we now present the optimal caching and delivery scheme for $z = \frac{K-1}{K\gamma}$ for any $K\gamma$.

5.4.1 Cache Placement Scheme

In the cache placement phase, each file $W^{(n)}$ is first split into K subfiles $W_{\phi}^{(n)}$, $\phi \in \Phi$ where each $K\gamma$-tuple $\phi \triangleq \{\phi_1, \phi_2, \ldots, \phi_{K\gamma}\}$ is drawn from the set

$$\Phi \triangleq \{\phi : \phi_1 \in [K], \phi_j = \langle \phi_{j-1} + z \rangle, \forall j \in [2 : K\gamma]\} \tag{5.18}$$

of size K. Then each cache k is filled as follows

$$\mathcal{Z}_k = \{W_{\Phi}^{(n)} \mid \forall n \in [N], \forall \Phi \ni k\} \tag{5.19}$$

forcing each integer $k \in [K]$ to appear in Φ exactly $K\gamma$ times, thus guaranteeing that each cache stores exactly $K\gamma$ subfiles from each file, thus respecting the cache-size constraint.

What the above placement also guarantees is that, by construction of the set Φ, all subfiles of each file stored in any z consecutive caches, are different. This is due to the fact that any two elements of each $K\gamma$-tuple $\phi \in \Phi$ have distance at least z. Thus, each user k has access to $K\gamma z = K\frac{K-1}{Kz}z = K-1$ different subfiles of its requested file W^{d_k}. We denote by $W_{\rho_k}^{(d_k)}$ the one remaining subfile desired by user k, for a specific $K\gamma$-tuple $\rho_k = \Phi \setminus \{\cup_{i\in\mathcal{C}_k} \mathcal{Z}_i\}$.

5.4.2 Delivery Scheme

Upon reception of the demand vector $\mathbf{d}$, the server multicasts a single XOR

$$X = \bigoplus_{k\in[K]} W_{\rho_k}^{(d_k)} \tag{5.20}$$

to all users of the network. By virtue of the fact that each user is only missing a single subfile, we can deduce that each user k can cache out from X all $K-1$ subfiles $\{W_{\rho_j}^{(d_j)}\}_{j\in[K]\setminus\{k\}}$ to successfully decode its own requested subfile $W_{\rho_k}^{(d_k)}$. Consequently the total delivery time is naturally equal to $T = |X| = \frac{1}{K}$.

5.4.3 Illustrative Example

Let us consider the case of $K\gamma = 3$, $z = 2$ and $K = 7$. We first split each file into 7 equally sized subfiles with indices

$$\Phi = \{135, 136, 146, 246, 247, 257, 357\}.$$

and fill each cache, according to (5.19), as follows

$$\begin{aligned}
\mathcal{Z}_1 &= \{W_{135}^{(n)}, W_{136}^{(n)}, W_{146}^{(n)},\ \forall n \in [N]\}\\
\mathcal{Z}_2 &= \{W_{246}^{(n)}, W_{247}^{(n)}, W_{257}^{(n)},\ \forall n \in [N]\}\\
\mathcal{Z}_3 &= \{W_{135}^{(n)}, W_{136}^{(n)}, W_{357}^{(n)},\ \forall n \in [N]\}\\
\mathcal{Z}_4 &= \{W_{146}^{(n)}, W_{246}^{(n)}, W_{247}^{(n)},\ \forall n \in [N]\}\\
\mathcal{Z}_5 &= \{W_{135}^{(n)}, W_{257}^{(n)}, W_{357}^{(n)},\ \forall n \in [N]\}\\
\mathcal{Z}_6 &= \{W_{136}^{(n)}, W_{146}^{(n)}, W_{246}^{(n)},\ \forall n \in [N]\}\\
\mathcal{Z}_7 &= \{W_{247}^{(n)}, W_{257}^{(n)}, W_{357}^{(n)},\ \forall n \in [N]\}.
\end{aligned}$$

In the delivery phase we consider the delivery vector $\mathbf{d} = (1, 2, \ldots, 7)$. We notice that the placement and topology jointly guarantee that each user is missing only a single subfile. For example, user 1 is only missing subfile $W_{357}^{(1)}$. Placing all these missing subfiles together, the server sends

$$X = W_{357}^{(1)} \oplus W_{146}^{(2)} \oplus W_{257}^{(3)} \oplus W_{136}^{(4)} \oplus W_{247}^{(5)} \oplus W_{135}^{(6)} \oplus W_{246}^{(7)} \tag{5.21}$$

which guarantees that each user can cache out exactly 6 elements to decode their own subfile. The delay is $T = \frac{1}{7}$ and the DoF of $K\gamma z + 1 = 7$.

5.5 Follow-Up Works

The topic of multi-access coded caching has recently attracted a lot of attention in the coded caching community. In this section we will briefly mention some of the new works on this topic. The scheme presented in [67] can be seen as a generalization of our results for the case $z = \frac{K-1}{K\gamma}$. The work in [68] proposes a novel transformation approach that nicely allow to convert schemes for the classical shared-link setting to the multi-access shared-link setting considered in this chapter. In [69] new index coding results were applied to the multi-access coded caching problem. A PDA-based scheme requiring lower subpacketization than known schemes was recently proposed in [70] for some limited values of the normalized cache size γ. In [71] the authors used cross resolvable designs to propose new multi-access coded caching scheme for a setting where the number of caches is different than the number of users in the system. Finally, the problem of secure delivery for multi-access cache-aided networks was addressed in [72], while schemes achieving demand privacy guarantees were presented in [73, 74].

Chapter 6

Novel Low-Complexity Scheme for the Cache-Aided MISO BC

In this chapter[1] we present a novel scheme for the MISO broadcast channel with users with dedicated caches, which has a low complexity both in terms of the subpacketization requirement and in the optimized beamforming design for the finite SNR regime. For the regime where the number of antennas N_0 is larger than the total normalized cache size t, our new algorithm is the first to achieve the exact one-shot linear optimal DoF with a subpacketization complexity that scales only linearly with the number of users. Knowing well that in the low-to-moderate SNR regimes beamforming gains can be as important as multiplexing gains, we proceed to consider the more general scenario where the multiplexing gain $\alpha \leq N_0$ is traded off with an ability to beamform in a manner that compensates the well-known effects of the worst-user channel condition. In our case, the multiplexing gain $\alpha \geq t$ is treated as a design parameter calibrated not only for yielding good finite-SNR performance but also for fine-tuning the subpacketization. The multicasting structure of our scheme allows for exploiting uplink-downlink duality in order to reduce the complexity design of optimized beamformers. In the end, our novel solution provides excellent performance for networks with finite SNR, finite file sizes, and many users.

6.1 The Subpacketization Requirement of Multi-Antenna Coded Caching Schemes

While the high subpacketization requirement is one of the major bottlenecks of shared-link coded caching schemes which struggle to achieve high gains in the finite file size regime, interesting results have emerged recently to significantly reduce the subpacketization of multi-antenna coded caching schemes. While the original schemes in [7, 8] required an

[1]The results in this chapter are part of a collaboration with a team from the University of Oulu and have produced the following publication [49], which is under revision for publication to Transaction on Wireless Communications. Before the beginning of the collaboration, the achievable scheme (for the DoF regime) denoted in this chapter by LIN, had been already published in [75].

astronomical subpacketization of $\binom{K}{t}\binom{K-t-1}{N_0-1}$, the recent work in [33] showed that if $\frac{K}{N_0}$ and $\frac{t}{N_0}$ are both integers, the optimal DoF $t+N_0$ is achievable with a subpacketization of $\binom{K/N_0}{t/N_0}$, which is dramatically less than the subpacketization in [2, 8]. This directly means that under fixed subpacketization constraints (fixed file size), adding multiple antennas can multiplicatively boost the real (subpacketization-constrained) DoF by a factor of N_0. The main idea behind the algorithm in [33], which after setting $\Lambda = \frac{K}{N_0}$ (as a by-product) coincides with the scheme in Section 3.3, is to employ user-grouping techniques to endow groups of users with the same cache content and then apply a specific precoding approach that decomposes the network of users into effectively parallel coded caching problems. While, as we know from Chapter 3, for single-antenna settings, having shared caches between the users causes an inevitable DoF loss, the work in [33] and our results of Chapter 3 have proven that multi-antenna shared-cache setups need not suffer from DoF losses. Of course, this is valid under the assumption that $\frac{K}{N_0}$ and $\frac{t}{N_0}$ are integers.[2]Another interesting work can be found in [76], which proposes a DoF-optimal scheme that yields a reduction in transmission and decoding complexity compared to the optimized beamformer scheme of [47], albeit with a small reduction in performance compared to [47], and also with an exponential subpacketization $\binom{K}{t}$. In another line of work, [48] provides a novel algorithm that reduces the channel state information (CSI) requirements, and does so with subpacketization $L_c\binom{K_c}{t}$, where $L_c \triangleq \frac{N_0+t}{t+1}$ and $K_c \triangleq \frac{K}{L_c}$. Finally, [77, 78] explore, under the assumption of $K = t + N_0$, how subpacketization can be traded-off with performance. Apart from our novel contributions in this chapter, the known existing multi-antenna schemes either exhibit subpacketization requirements that are exponential in K, or do not experience DoF optimality in scenarios where $N_0 > t$.

6.2 System Model and Performance Measure

We consider a MISO broadcast setting, in which a single server, equipped with N_0 transmit antennas, communicates with K single-antenna receiving users over a shared wireless link. An illustration of the considered communication setup for a small network of $K = 3$ users is provided in Figure 6.1. The server has access to a library $\mathcal{F}$ of $N \geq K$ files, where each file $W^{(n)}, n \in [N]$ has a size of f bits[3]. We assume that every user has a cache memory of size Mf bits. As usual, we use $t \triangleq \frac{KMf}{Nf}$ to denote the total cache size in the network normalized by the size of the library.

During the cache placement phase, the placement algorithm operates without any prior knowledge of future requests. We use $\mathcal{Z}_k$ to denote the content cached at user $k \in [K]$. At the beginning of the delivery phase, each user $k \in [K]$ reveals its requested file $W^{(d_k)}$ to the server, which, after receiving the demand vector $\mathbf{d} = (d_1, d_2, \ldots, d_K)$, follows the delivery algorithm to transmit the requested subpackets to the users. This

[2]The scheme of [33] suffers DoF losses (and also increased subpacketization) if either $\frac{K}{N_0}$ or $\frac{t}{N_0}$ is non-integer. The DoF is reduced by a multiplicative factor that can reach 2 when $N_0 > t$, and can reach $\frac{3}{2}$ when $N_0 < t$.

[3]While in all other chapters we assume that the size of each file is normalized by the total number of bits, so that they are unit-sized, in this chapter we highlight the composition of each file as a sequence of bits in order to work with the transmission rate as a performance metric.

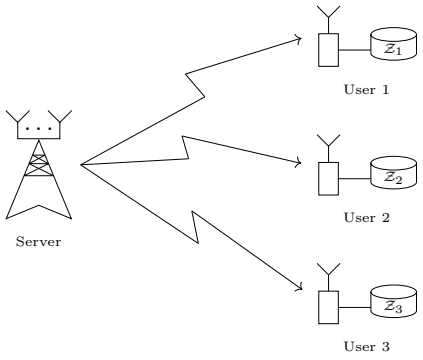


Figure 6.1 – Illustration of the communication setting for a network with $K = 3$ users

will involve the transmission of some I transmission vectors $\{\mathbf{x}_i\} \in \mathbb{C}^{N_0}$, $i \in [I]$, where I is given by the delivery algorithm. These transmission vectors are transmitted in consecutive time intervals or separate frequency bins[4], using the array of N_0 antennas. After $\mathbf{x}_i$ is transmitted, user k receives $y_i(k) = \mathbf{h}_k^H \mathbf{x}_i + w_i(k)$, where $\mathbf{h}_k \in \mathbb{C}^{N_0}$ denotes the channel vector and $w_i(k) \sim \mathbb{C}\mathcal{N}(0, n_0)$ denotes the observed noise at user k. Furthermore, we consider a slow-fading model in which the channel vectors remain constant during each time interval i, and we assume that full channel state information (CSI) is available at the server.[5]

Let $\mathcal{X}_i \subseteq [K]$ denote the set of users targeted by $\mathbf{x}_i$, and let T_i denote the duration of time interval i required so that every user in $\mathcal{X}_i$ decodes its intended data from $\mathbf{x}_i$. If we consider B_i to be the length of the codeword transmitted at time slot i, and if we define R_i to be the multicast rate at which the server transmits a common message to all users in $\mathcal{X}_i$, then T_i is simply the ratio between B_i and R_i. We will use the metric of the *symmetric rate*, which describes the total number of bits per second with which each user is served. Particularly, we will consider the worst-case metric, corresponding to the symmetric rate at which the system can serve all users in the network irrespective of the demand vector $\mathbf{d}$. Given that the delivery phase has an overall duration of $\sum_{i=1}^{I} T_i$, and given that there are K users, the symmetric rate can be computed as

$$R_{sym} = \frac{Kf}{\sum_{i=1}^{I} T_i} = \frac{Kf}{\sum_{i=1}^{I} \frac{B_i}{R_i}}. \tag{6.1}$$

Our aim is to design a placement and delivery scheme that maximizes R_{sym}.

[4]For comparison with other schemes, we will generally assume that transmissions here are made in consecutive time intervals.

[5]Improving CSI accuracy is a well-studied topic in the literature. In general, the designs are specific to the underlying uncertainty model, i.e., whether CSI error is bounded [79] or unbounded [80]. If the error is bounded, robust (worst-case) beamforming solutions can be computed that guarantee the achievability of the max-min SINR for all possible realizations of channel uncertainty. For unbounded error, statistically robust solutions can be provided if the estimation noise is assumed to follow a known distribution. In such a scenario, fixed-point iterations are used to provide a robust solution by adding the noise contribution from CSI uncertainty to the thermal noise.

6.2.1 Building the Transmission Vectors

We use linear precoding to build the transmission vectors. A generic transmission vector $\mathbf{x}_i$ is built as

$$\mathbf{x}_i = \sum_{k \in \mathcal{X}_i} \mathbf{w}_i(k) X_i(k) \ , \tag{6.2}$$

where $X_i(k)$ is the data codeword transmitted to user k, and $\mathbf{w}_i(k) \in \mathbb{C}^{N_0}$ is the beamforming vector used for $X_i(k)$. The beamforming vectors $\mathbf{w}_i(k)$ are here designed to maximize the worst-user rate, or equivalently, the worst user's SINR (signal to interference and noise ratio), as $\mathbf{x}_i$ is transmitted. Thus, given the transmission model in (6.2), the multicast rate at time interval i is calculated as

$$R_i = \log(1 + \omega_i^*) \ , \tag{6.3}$$

in which ω_i^* is defined as

$$\omega_i^* = \max_{\mathbf{w}_i(k)} \min_{k \in \mathcal{X}_i} \text{SINR}_k \quad s.t. \quad \sum_{k \in \mathcal{X}_i} ||\mathbf{w}_i(k)||^2 \leq P_T \ , \tag{6.4}$$

where SINR_k is the received SINR at user k and P_T is the available transmission power. We discuss this optimization problem in more details in the next section.

As suggested before, we will consider that each transmission vector serves $|\mathcal{X}_i| = t + \alpha$ users, where $\alpha \leq N_0$ is the multiplexing gain and is treated as a parameter of choice that can be tuned to obtain a better rate performance at finite-SNR. This α represents the number of independent streams in each transmission, and hence, reducing it implies sacrificing some spatial multiplexing gain for the purpose of increasing the beamforming gain, which can help reduce the worst-user effect in the finite-SNR regime. As we discuss later on, this same α can also be calibrated to control subpacketization as well as beamformer design complexity.

6.3 A Cyclic Caching Scheme for Reduced Subpacketization

In this section, we present our low-complexity high performance cyclic caching scheme, which can be applied to any MISO setup in which $\alpha \geq t$.[6] The following theorem summarizes the DoF and subpacketization performance of the scheme:

Theorem 9. *For the large MISO broadcast setup with $t \leq \alpha \leq N_0$, the sum DoF of $t + \alpha$ is achievable with a subpacketization*[7]

$$\frac{K(t+\alpha)}{\big(gcd(K,t,\alpha)\big)^2} \ , \tag{6.5}$$

[6] For setups with $t > \alpha$, one can use the coded caching scheme presented in [33] for reduced subpacketization.

[7] Here, $gcd(K,t,\alpha)$ corresponds to the greatest common divisor of K, t, α.

Proof. The proof is found in this current section, where we present the designed cyclic caching scheme and show that it employs the above subpacketization to achieve the sum DoF of $t + \alpha$. □

In what follows, we first introduce a cache placement algorithm in Section 6.3.1, which is based on a well-defined placement matrix and requires each file to be split into $K(t + \alpha)$ subfiles. In Sections 6.3.2 and 6.3.3, we explain the delivery phase, in which the missing subfiles are delivered to the requesting users with $I = K(K - t)$ multicast transmission vectors, each serving $t + \alpha$ subpackets to $t + \alpha$ different users. In Section 6.3.4, using an example network, we show that this delivery algorithm follows a simple graphical representation, involving circular shifting of two vectors over a tabular structure. Overall, in Sections 6.3.1 to 6.3.4, we present a scheme which satisfies all the requests with multicast transmissions always containing $t + \alpha$ subfiles, which implies a DoF of $t + \alpha$ with subpacketization $K(t + \alpha)$. Finally, in Section 6.3.6 we show that by properly applying a user-grouping technique, subpacketization is further reduced by a factor of $\big(gcd(K, t, \alpha)\big)^2$, without any DoF loss.

Remark 8. *When $\alpha = N_0$, the achieved DoF $t+N_0$ is exactly optimal under the assumption of one-shot linear schemes and uncoded placement (cf. [22]).*

We note that, for fixed t and N_0, the above integer subpacketization scales linearly with K. This allows applying coded caching in larger networks, and entails the benefit of a reduced number of necessary transmissions which in turn implies a reduced number of beamformer design problems that need to be solved. As a quick comparison, if $K = 20, t = 4, N_0 = \alpha = 8$, the proposed scheme requires subpacketization of 15, while the schemes in [76] and [8] respectively require (approximately) 5×10^3 and 3×10^7 subpackets. More comparisons are provided in Section 6.4.2.

Remark 9. *In Theorem 9, the term $gcd(K, t, \alpha)$ represents the number of users that will store the same content in their caches. Therefore, in the delivery phase, this dedicated-cache setting can be seen as a shared-cache setting with $\Lambda = \frac{K}{gcd(K,t,\alpha)}$ caches and $gcd(K, t, \alpha)$ users per cache.*

Before proceeding with the description of the algorithm, we introduce some useful notation. We use $[i : j]$ to represent the vector $[i\ i{+}1\ ...\ j]$. $\mathbf{V}[i, j]$ refers to the element at the i-th row and j-th column of matrix $\mathbf{V}$, and $\mathbf{w}[i]$ represents the i-th element in vector $\mathbf{w}$.

6.3.1 Cache Placement

For cache placement, we use a $K \times K$ binary placement matrix $\mathbf{V}$ where the first row has t consecutive 1's (other elements are zero) and each subsequent row is a circular shift of the previous row by one column. Given $\mathbf{V}$, we split each file $W^{(n)}, n \in [N]$ into K subfiles $W_p^{(n)}$, $p \in [K]$, and each subfile $W_p^{(n)}$ into $t + \alpha$ smaller minifiles $W_{p,q}^{(n)}$.

Then for every $p, k \in [K]$, if $\mathbf{V}[p,k] = 1$, $W_{p,q}^{(n)}$ is stored in the cache memory of user k, $\forall n \in [N], q \in [t+\alpha]$.[8]

Example 2. *For a scenario of $K = 6$, $t = 2$, $\alpha = 3$, $\mathbf{V}$ is built as*

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} , \tag{6.6}$$

and the resulting subpacketization is $K \times (t+\alpha) = 6 \times (2+3) = 30$. For example, the cache contents of users 1 and 2 can be found from (6.6) as

$$\mathcal{Z}_1 = \{W_{1,q}^{(n)}, W_{6,q}^{(n)} \; ; \; \forall\, n \in [N],\, q \in [5]\} \; , \quad \mathcal{Z}_2 = \{W_{1,q}^{(n)}, W_{2,q}^{(n)} \; ; \; \forall\, n \in [N],\, q \in [5]\} \; .$$

The cache contents of users $3-6$ can be written accordingly.

6.3.2 Content Delivery

In cyclic caching, the content delivery phase consists of K rounds, where in each round we build $K - t$ transmission vectors. Thus, the content delivery is completed after $I = K(K-t)$ transmissions. We use $\mathbf{x}_j^r$ to denote the transmission vector $j \in [K-t]$ at transmission round $r \in [K]$.[9] By transmitting $\mathbf{x}_j^r$, useful subfiles are delivered to a set of $t+\alpha$ users. We define the user index vector $\mathbf{k}_j^r$ to denote the set of users being targeted by $\mathbf{x}_j^r$, and the subfile index vector $\mathbf{p}_j^r$ to contain the subfiles indices targeted for the users in $\mathbf{k}_j^r$. In other words, using $\mathbf{x}_j^r$, we transmit (part of) the subfile $W_{\mathbf{p}_j^r[n]}^{(d_{\mathbf{k}_j^r[n]})}$ to each user $\mathbf{k}_j^r[n]$, $n = 1, \ldots, t+\alpha$. Both $\mathbf{k}_j^r$ and $\mathbf{p}_j^r$ vectors are built recursively. Let us use $\%$ sign to denote the mod operator with an offset of one. It is defined as $a\%b = ((a-1) \bmod b) + 1$, such that $a\%a = a$ and $(a+b)\%a = b\%a$. Then, $\mathbf{k}_j^1$ and $\mathbf{p}_j^1$ are built as

$$\begin{aligned} \mathbf{k}_j^1 &= \Big[\; [1:t] \parallel \big(([1:\alpha] + j - 1)\%(K-t)\big) + t \;\Big] \; , \\ \mathbf{p}_j^1 &= \Big[\big((t + j - [1:t])\%(K-t)\big) + [1:t] \parallel \mathbf{e}(\alpha)\Big] \; , \end{aligned} \tag{6.7}$$

where $\mathbf{e}(m)$ is a vector of 1's with size m (e.g., $\mathbf{e}(3) = [1\ 1\ 1]$). For the next transmission rounds, i.e., $1 < r \leq K$, we simply build $\mathbf{k}_j^r$ and $\mathbf{p}_j^r$, using $\mathbf{k}_j^1$ and $\mathbf{p}_j^1$, as

$$\mathbf{k}_j^r = \big(\mathbf{k}_j^1 + r\big)\%K \; , \quad \mathbf{p}_j^r = \big(\mathbf{p}_j^1 + r\big)\%K \; . \tag{6.8}$$

To gain a better insight into how $\mathbf{k}_j^r$ and $\mathbf{p}_j^r$ are built, in Section 6.3.4, we offer a simple graphical representation, which is based on circular shift operations over a tabular structure. In the following, we provide $\mathbf{k}_j^r$ and $\mathbf{p}_j^r$ vectors for the small network scenario given in Example 2.

[8] The placement matrix $\mathbf{V}$ used in this paper is a special case of valid placement matrices introduced in [77].

[9] In the general transmission vector model (6.2), $\mathbf{x}_j^r$ corresponds to $\mathbf{x}_i$, $i = (r-1)(K-t) + j$.

Example 3. *In the scenario of Example 2, content delivery consists of six rounds, where at each round four transmission vectors are built. The user and subfile index vectors for the first and second rounds are given as*

$$\begin{aligned} &\mathbf{k}_1^1 = [1\ 2\ 3\ 4\ 5]\ , \quad \mathbf{k}_2^1 = [1\ 2\ 4\ 5\ 6]\ , \quad \mathbf{k}_3^1 = [1\ 2\ 5\ 6\ 3]\ , \quad \mathbf{k}_4^1 = [1\ 2\ 6\ 3\ 4]\ , \\ &\mathbf{p}_1^1 = [3\ 3\ 1\ 1\ 1]\ , \quad \mathbf{p}_2^1 = [4\ 4\ 1\ 1\ 1]\ , \quad \mathbf{p}_3^1 = [5\ 5\ 1\ 1\ 1]\ , \quad \mathbf{p}_4^1 = [2\ 6\ 1\ 1\ 1]\ , \end{aligned} \tag{6.9}$$

and

$$\begin{aligned} &\mathbf{k}_1^2 = [2\ 3\ 4\ 5\ 6]\ , \quad \mathbf{k}_2^2 = [2\ 3\ 5\ 6\ 1]\ , \quad \mathbf{k}_3^2 = [2\ 3\ 6\ 1\ 4]\ , \quad \mathbf{k}_4^2 = [2\ 3\ 1\ 4\ 5]\ , \\ &\mathbf{p}_1^2 = [4\ 4\ 2\ 2\ 2]\ , \quad \mathbf{p}_2^2 = [5\ 5\ 2\ 2\ 2]\ , \quad \mathbf{p}_3^2 = [6\ 6\ 2\ 2\ 2]\ , \quad \mathbf{p}_4^2 = [3\ 1\ 2\ 2\ 2]\ , \end{aligned} \tag{6.10}$$

respectively. The user and subfile index vectors for the other rounds are built similarly.

Two other variables are needed to build the transmission vector $\mathbf{x}_j^r$. First, we introduce the minifile index $q(n,p)$, where $n \in [N]$ refers to a general file and $p \in [K]$ is the subfile index. The minifile index $q(n,p)$ indicates which minifile of $W_p^{(n)}$ should be transmitted, the next time it is included in a transmission vector. For every $n \in [N]$ and $p \in [K]$, $q(n,p)$ is initialized to one, and incremented every time $W_p^{(n)}$ is included in a transmission vector. For notational simplicity, here we use

$$q_j^r(n) \triangleq q(d_{\mathbf{k}_j^r[n]}, \mathbf{p}_j^r[n])\ . \tag{6.11}$$

Second, we use the *interference indicator* set $\mathcal{R}_j^r(n)$ to be the set of users at which $W^{(d_{\mathbf{k}_j^r[n]})}_{\mathbf{p}_j^r[n],q_j^r(n)}$ should be suppressed by beamforming.[10] $\mathcal{R}_j^r(n)$ has exactly $\alpha - 1$ elements and is built as

$$\mathcal{R}_j^r(n) = \Big\{ k \in \mathbf{k}_j^r \backslash \mathbf{k}_j^r[n] \quad | \quad \mathbf{V}\big[\mathbf{p}_j^r[n], k\big] = 0 \Big\}\ . \tag{6.12}$$

Example 4. *For the network considered in Examples 2 and 3, the interference indicator sets for the first transmission round are built as*

$$\begin{aligned} \mathcal{R}_1^1(1) = \{2,5\}, \quad \mathcal{R}_1^1(2) = \{1,5\}, \quad \mathcal{R}_1^1(3) = \{4,5\}, \\ \mathcal{R}_1^1(4) = \{3,5\}, \quad \mathcal{R}_1^1(5) = \{4,4\}. \end{aligned} \tag{6.13}$$

Finally, the transmission vectors are built as:[11]

$$\mathbf{x}_j^r = \sum_{n=1}^{t+\alpha} \mathbf{w}_{\mathcal{R}_j^r(n)} W^{(d_{\mathbf{k}_j^r[n]})}_{\mathbf{p}_j^r[n],q_j^r(n)}\ . \tag{6.14}$$

[10] If zero-forcing beamformers are used, $\mathcal{R}_j^r(n)$ denotes the set of users at which $W^{(d_{\mathbf{k}_j^r[n]})}_{\mathbf{p}_j^r[n],q_j^r(n)}$ should be nulled-out.

[11] The general transmission vector model in (6.2) is equivalent to (6.14) via the following index mapping:

$$k \to \mathbf{k}_j^r[n]\ , \qquad \mathbf{w}_i \to \mathbf{w}_{\mathcal{R}_j^r(n)}\ , \quad \mathcal{X}_i \to \bigcup_{n\in[t+\alpha]} \{\mathbf{k}_j^r[n]\}\ , \quad X_i(k) \to W^{(d_{\mathbf{k}_j^r[n]})}_{\mathbf{p}_j^r[n],q_j^r(n)}\ .$$

6.3.3 Decoding at the Receiver

During time interval i, every user $k \in \mathcal{X}_i$ receives

$$y_i(k) = \mathbf{h}_k^H \mathbf{w}_i(k) X_i(k) + \sum_{\hat{k} \in \mathcal{X}_i \setminus \{k\}} \mathbf{h}_k^H \mathbf{w}_i(\hat{k}) X_i(\hat{k}) + w_i(k) \, , \tag{6.15}$$

where the first term is the intended codeword and the latter two terms indicate the interference and noise, respectively. Assume $\hat{k} \triangleq \mathbf{k}_{\hat{j}}^{\hat{r}}[\hat{n}]$, for some $\hat{j}, \hat{r}, \hat{n}$. Defining $p(\hat{k}) \triangleq \mathbf{p}_{\hat{j}}^{\hat{r}}[\hat{n}]$, for every element in the interference term only one of the following options is possible:

1. $\mathbf{V}[p(\hat{k}), k] = 1$ indicates $X_i(\hat{k})$ is in the cache memory of user k, and hence, $\mathbf{h}_k^H \mathbf{w}_i(\hat{k}) X_i(\hat{k})$ can be reconstructed and removed from $y_i(k)$;

2. $\mathbf{V}[p(\hat{k}), k] = 0$ indicates that $\hat{k}$ is in the interference indicator set associated with $X_i(k)$ as defined in (6.12), and hence, $X_i(\hat{k})$ is suppressed at user k by transmit beamforming.

In both cases, the interference due to $X_i(\hat{k})$ can be controlled and/or completely removed at user k. Since $|\mathcal{X}_i| = t + \alpha$, the proposed scheme allows for serving $t + \alpha$ users in parallel during each transmission interval. The following example clarifies the decoding procedure for a single transmission in a small network. A more detailed explanation is provided in Appendix C.1.

Example 5. *Consider the network in Example 2, for which the user and subfile index vectors are provided in Example 3 and the interference indicator sets are presented in Example 4. Let us assume that users* $(1, 2, \ldots, 6)$ *request files* (A, B, C, D, E, F)*, respectively. Then, following* (6.14)*, the first transmission vector in the first round is built as*

$$\mathbf{x}_1^1 = \mathbf{w}_{2,5} A_{3,1} + \mathbf{w}_{1,5} B_{3,1} + \mathbf{w}_{4,5} C_{1,1} + \mathbf{w}_{3,5} D_{1,1} + \mathbf{w}_{3,4} E_{1,1} \, , \tag{6.16}$$

where the brackets of the interference indicator sets are dropped for notation simplicity. After $\mathbf{x}_1^1$ *is transmitted, user 1 receives*

$$y_1^1(1) = \mathbf{h}_1^H \mathbf{w}_{2,5} A_{3,1} + \underline{\underline{\mathbf{h}_1^H \mathbf{w}_{1,5} B_{3,1}}} + \underline{\mathbf{h}_1^H \mathbf{w}_{4,5} C_{1,1}} + \underline{\mathbf{h}_1^H \mathbf{w}_{3,5} D_{1,1}} + \underline{\mathbf{h}_1^H \mathbf{w}_{3,4} E_{1,1}} + w_1^1(1) \, , \tag{6.17}$$

in which the single- and double-underlined terms indicate the interference. From Example 2, $C_{1,1}$*,* $D_{1,1}$*, and* $E_{1,1}$ *are available in the cache memory of user 1, and hence, all the single-underlined terms can be reconstructed and removed from the received signal. On the other hand, the minifile* $B_{3,1}$ *requested by user 2 is suppressed at user 1 via transmit beamforming, following the definition of the interference indicator sets, the double-underlined term is also suppressed at user 1 with the help of the beamforming vectors. As a result, user 1 can decode* $A_{3,1}$ *with controlled interference. Similarly, users* $2-5$ *can decode* $B_{3,1}$*,* $C_{1,1}$*,* $D_{1,1}$ *and* $E_{1,1}$*, respectively. In Table 6.1, we have summarized how different users decode* $\mathbf{x}_1^1$ *and extract their requested data.*

Transmission vector: $\mathbf{x}_1^1 = \mathbf{w}_{2,5}A_{3,1} + \mathbf{w}_{1,5}B_{3,1} + \mathbf{w}_{4,5}C_{1,1} + \mathbf{w}_{3,5}D_{1,1} + \mathbf{w}_{3,4}E_{1,1}$

User	Avail. in cache	Supp. by beamformer	Useful data	SINR
1	$C_{1,1}, D_{1,1}, E_{1,1}$	$B_{3,1}$	$A_{3,1}$	$\frac{\lvert\mathbf{h}_1^H\mathbf{w}_{2,5}\rvert^2}{\lvert\mathbf{h}_1^H\mathbf{w}_{1,5}\rvert^2+n_0}$
2	$C_{1,1}, D_{1,1}, E_{1,1}$	$A_{3,1}$	$B_{3,1}$	$\frac{\lvert\mathbf{h}_1 2^H\mathbf{w}_{1,5}\rvert^2}{\lvert\mathbf{h}_2^H\mathbf{w}_{2,5}\rvert^2+n_0}$
3	$A_{3,1}, B_{3,1}$	$D_{1,1}, E_{1,1}$	$C_{1,1}$	$\frac{\lvert\mathbf{h}_3^H\mathbf{w}_{4,5}\rvert^2}{\lvert\mathbf{h}_3^H\mathbf{w}_{3,5}\rvert^2+\lvert\mathbf{h}_3^H\mathbf{w}_{3,4}\rvert^2+n_0}$
4	$A_{3,1}, B_{3,1}$	$C_{1,1}, E_{1,1}$	$D_{1,1}$	$\frac{\lvert\mathbf{h}_4^H\mathbf{w}_{3,5}\rvert^2}{\lvert\mathbf{h}_4^H\mathbf{w}_{4,5}\rvert^2+\lvert\mathbf{h}_4^H\mathbf{w}_{3,4}\rvert^2+n_0}$
5	$A_{3,1}, B_{3,1}$	$C_{1,1}, D_{1,1}$	$E_{1,1}$	$\frac{\lvert\mathbf{h}_5^H\mathbf{w}_{3,4}\rvert^2}{\lvert\mathbf{h}_5^H\mathbf{w}_{4,5}\rvert^2+\lvert\mathbf{h}_5^H\mathbf{w}_{3,5}\rvert^2+n_0}$
6	–	–	–	–

Table 6.1 – Decoding process for $\mathbf{x}_1^1$ at different network users, for Example 5.

6.3.4 A Graphical Example

For further clarification, we describe the operation of the cyclic caching scheme for the network setup in Example 2, using a graphical representation of the placement matrix $\mathbf{V}$ in Figure 6.2. In this figure, each column represents a user, and each row denotes a subfile index. For example, the first column represents user one, and the first row stands for the first subfile of all files, i.e., $W_{1,q}^{(n)}$, $\forall n \in [N], q \in [t+\alpha]$. Lightly shaded entries indicate subfiles that are cached at the user. For example, $W_{1,q}^{(n)}, W_{6,q}^{(n)}$ are stored at user 1, $\forall n \in [N], q \in [t+\alpha]$.

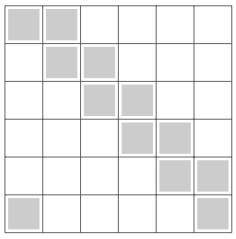
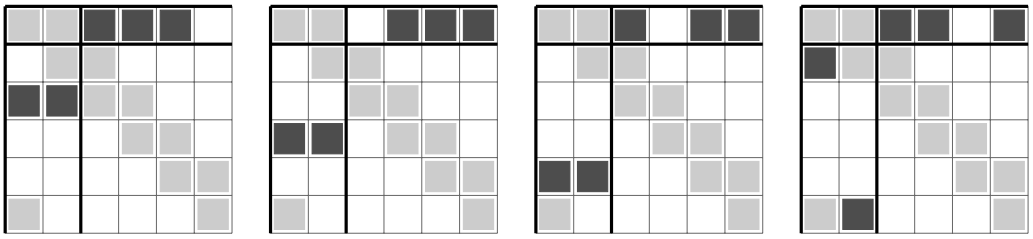


Figure 6.2 – Graphical illustration for Example 1.

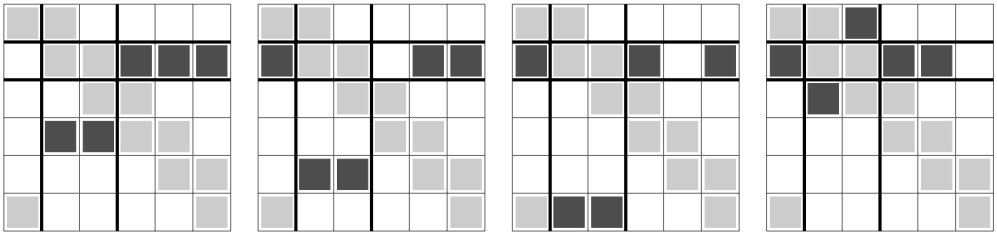
In the subsequent figures, we use darkly shaded entries to indicate which subfile indices of the requested files are sent during each transmission. The column and row indices of these darkly shaded entries are extracted from the user and subfile index vectors. For our example network, the user and subfile index vectors for the first and second transmission rounds are provided in (6.9) and (6.10); and their graphical representations are depicted in Figures 6.3 and 6.4. For example, Fig. 6.3a corresponds to the first transmission of the first round, where users $\mathbf{k}_1^1 = [1,2,3,4,5]$ receive minifiles of subfiles indicated by $\mathbf{p}_1^1 = [3,3,1,1,1]$. For simplicity, let us assume that users request files (A,B,C,D,E,F). Then, users 1-5 receive $A_{3,1}$, $B_{3,1}$, $C_{1,1}$, $D_{1,1}$ and $E_{1,1}$, respectively.

The transmission vectors can be easily reconstructed using the graphical representations. For example, Fig. 6.3 implies that the following transmission vectors are generated



(a) Transmission 1 (b) Transmission 2 (c) Transmission 3 (d) Transmission 4

Figure 6.3 – Graphical illustration of the first round $r = 1$.



(a) Transmission 1 (b) Transmission 2 (c) Transmission 3 (d) Transmission 4

Figure 6.4 – Graphical illustration of the second round $r = 2$.

in the first round:

$$\begin{aligned}
\mathbf{x}_1^1 =& \mathbf{w}_{2,5}A_{3,1} + \mathbf{w}_{1,5}B_{3,1} + \mathbf{w}_{4,5}C_{1,1} + \mathbf{w}_{3,5}D_{1,1} + \mathbf{w}_{3,4}E_{1,1} \ , \\
\mathbf{x}_2^1 =& \mathbf{w}_{2,6}A_{4,1} + \mathbf{w}_{1,6}B_{4,1} + \mathbf{w}_{5,6}D_{1,2} + \mathbf{w}_{4,6}E_{1,2} + \mathbf{w}_{4,5}F_{1,1} \ , \\
\mathbf{x}_3^1 =& \mathbf{w}_{2,3}A_{5,1} + \mathbf{w}_{1,3}B_{5,1} + \mathbf{w}_{6,3}E_{1,3} + \mathbf{w}_{5,3}F_{1,2} + \mathbf{w}_{5,6}C_{1,2} \ , \\
\mathbf{x}_4^1 =& \mathbf{w}_{4,6}A_{2,1} + \mathbf{w}_{3,4}B_{6,1} + \mathbf{w}_{3,4}F_{1,3} + \mathbf{w}_{6,4}C_{1,3} + \mathbf{w}_{6,3}D_{1,3} \ ,
\end{aligned}$$

where the brackets of the interference indicator sets are dropped for notation simplicity. Note that according to (6.11), the minifile index $q_j^1(n)$ is equivalent to $q(d_{\mathbf{k}_j^1[n]}, \mathbf{p}_j^1[n])$, and hence, is incremented every time $W_{\mathbf{p}_j^1[n]}^{(d_{\mathbf{k}_j^1[n]})}$ appears in a transmission vector. As a result, the minifile index for the subfile C_1 is incremented from one to three, as it has appeared in $\mathbf{x}_1^1$, $\mathbf{x}_3^1$ and $\mathbf{x}_4^1$, respectively.

Following the same procedure, using Figure 6.4, for the second round we have

$$\begin{aligned}
\mathbf{x}_1^2 =& \mathbf{w}_{3,6}B_{4,2} + \mathbf{w}_{2,6}C_{4,1} + \mathbf{w}_{5,6}D_{2,1} + \mathbf{w}_{4,6}E_{2,1} + \mathbf{w}_{4,5}F_{2,1} \ , \\
\mathbf{x}_2^2 =& \mathbf{w}_{3,1}B_{5,2} + \mathbf{w}_{2,1}C_{5,1} + \mathbf{w}_{6,1}E_{2,2} + \mathbf{w}_{5,1}F_{2,2} + \mathbf{w}_{5,6}A_{2,2} \ , \\
\mathbf{x}_3^2 =& \mathbf{w}_{3,4}B_{6,2} + \mathbf{w}_{2,4}C_{6,1} + \mathbf{w}_{1,4}F_{2,3} + \mathbf{w}_{6,4}A_{2,3} + \mathbf{w}_{6,1}D_{2,2} \ , \\
\mathbf{x}_4^2 =& \mathbf{w}_{5,1}B_{3,2} + \mathbf{w}_{4,5}C_{1,4} + \mathbf{w}_{4,5}A_{2,4} + \mathbf{w}_{1,5}D_{2,3} + \mathbf{w}_{1,4}E_{2,3} \ .
\end{aligned}$$

From Figures 6.3 and 6.4, it can be seen that the transmissions vectors $\mathbf{x}_2^r$, $\mathbf{x}_3^r$ and $\mathbf{x}_4^r$ in round r are built by circular shift of the first transmission vector $\mathbf{x}_1^r$ over the non-shaded cells of the tabular grid and in two perpendicular directions. Specifically, the first two

terms in $\mathbf{x}_1^r$ are shifted vertically, while the other three are shifted horizontally. This procedure is highlighted in sub-figures using wide border lines. Moreover, comparing Figures 6.3 and 6.4, it is clear that the vectors in the second transmission round are diagonally shifted versions of the vectors in the first round. This property is the intuition behind the cyclic caching name, and results from the recursive procedure in (6.8), where the mod operator is used to build $\mathbf{k}_j^r$ and $\mathbf{p}_j^r$ vectors for $r > 1$.

6.3.5 Beamformer Design

As discussed earlier, we use optimized beamformers to build the transmission vectors. These beamformers result in a better rate compared to zero-forcing, especially in the low-SNR regime, as they allow balancing the detrimental impact of noise and the inter-stream interference [47]. However, optimized beamformers may require non-convex optimization problems to be solved (due to interference from unwanted terms), making the problem computationally intractable even for moderate K values. Interestingly, cyclic caching, in addition to requiring much-reduced subpacketization, also manages to eliminate the requirement of multicasting, thus enabling optimized beamformers to be designed with much less computational complexity.

As $t + \alpha$ users are served simultaneously by each transmission vector, symmetric rate maximization is equivalent to maximizing the worst user rate (among served users), which, in turn, is equivalent to maximizing the worst user SINR. Naturally, the unwanted terms canceled-out using the local cache contents are *not* considered as interference in the optimized SINR expressions. The optimized beamformer vectors for the j-th transmission in round r can be found by solving the optimization problem

$$\max_{\mathbf{w}_{\mathcal{R}_j^r(n)}} \min_{n\in[t+\alpha]} \frac{|\mathbf{h}_{\mathbf{k}_j^r[n]}^H \mathbf{w}_{\mathcal{R}_j^r(n)}|^2}{\sum\limits_{b\,:\,\mathcal{R}_j^r[b]\ni \mathbf{k}_j^r[n]} |\mathbf{h}_{\mathbf{k}_j^r[n]}^H \mathbf{w}_{\mathcal{R}_j^r(b)}|^2 + n_0} \qquad s.t. \quad \sum_{n\in[t+\alpha]} |\mathbf{w}_{\mathcal{R}_j^r(n)}|^2 \le P_T \ . \tag{6.18}$$

Example 6. *Consider the network in Example 2 and the transmitted signal vector* $\mathbf{x}_1^1$ *in* (6.16) *for the first transmission of the round* $r = 1$*. The optimized beamformers* $\mathbf{w}_{25}, \mathbf{w}_{15}, \mathbf{w}_{45}, \mathbf{w}_{35}, \mathbf{w}_{34}$ *can be found by solving*

$$\begin{aligned} &\max_{\mathbf{w}_{\mathcal{R}}} \ \min\Big\{\frac{|\mathbf{h}_1^H\mathbf{w}_{25}|^2}{\mu_1+n_0}, \frac{|\mathbf{h}_2^H\mathbf{w}_{15}|^2}{\mu_2+n_0}, \frac{|\mathbf{h}_3^H\mathbf{w}_{45}|^2}{\mu_3+n_0}, \frac{|\mathbf{h}_4^H\mathbf{w}_{35}|^2}{\mu_4+n_0}, \frac{|\mathbf{h}_5^H\mathbf{w}_{34}|^2}{\mu_5+n_0}\Big\} \\ &s.t. \ \ |\mathbf{w}_{25}|^2 + |\mathbf{w}_{15}|^2 + |\mathbf{w}_{45}|^2 + |\mathbf{w}_{35}|^2 + |\mathbf{w}_{34}|^2 \le P_T, \end{aligned}$$

where μ_k *denotes the interference at user* k*, given as*

$$\begin{aligned} &\mu_1 = |\mathbf{h}_1^H\mathbf{w}_{15}|^2, \quad \mu_2 = |\mathbf{h}_2^H\mathbf{w}_{25}|^2, \quad \mu_3 = |\mathbf{h}_3^H\mathbf{w}_{35}|^2 + |\mathbf{h}_3^H\mathbf{w}_{34}|^2, \\ &\mu_4 = |\mathbf{h}_4^H\mathbf{w}_{34}|^2 + |\mathbf{h}_4^H\mathbf{w}_{45}|^2, \quad \mu_5 = |\mathbf{h}_5^H\mathbf{w}_{15}|^2 + |\mathbf{h}_5^H\mathbf{w}_{25}|^2 + |\mathbf{h}_5^H\mathbf{w}_{35}|^2 + |\mathbf{h}_5^H\mathbf{w}_{45}|^2. \end{aligned}$$

In cyclic caching, as also demonstrated in Example 6, the number of interfering messages experienced by each user does not need to be the same, in general. For the transmission vector $\mathbf{x}_1^1$ considered in Example 6, users 1-5 experience 1, 1, 2, 2, and 4

interfering messages, respectively. This unevenness is an intrinsic characteristic of the proposed cyclic caching scheme, while each message is still suppressed at exactly $\alpha - 1$ users (in Example 6, there exist exactly $\alpha - 1 = 2$ users in each interference indicator set).[12]

The optimized beamformer design problems tend to be non-convex in general and require iterative methods such as successive convex approximation (SCA) to be used [47]. Such methods can be computationally complex and make the implementation infeasible, especially for large networks. However, the special unicasting nature of the cyclic caching transmission and the max-min SINR objective in (6.18) make the optimization problem quasi-convex [81, 82], and hence, allow us to use uplink-downlink duality to attain a simple iterative solution. As the beamformer design with uplink-downlink duality is thoroughly discussed in [82], here we only briefly review the required process. Denoting the normalized receive beamforming vectors for the dual uplink channel as $\mathbf{v}_{\mathcal{R}_j^r(n)}$, $r \in [K]$ and $j \in [K-t]$, the uplink-downlink duality necessitates

$$\sum_{n\in[t+\alpha]} \nu_n \|\mathbf{v}_{\mathcal{R}_j^r(n)}\|^2 = \sum_{n\in[t+\alpha]} \|\mathbf{w}_{\mathcal{R}_j^r(n)}\|^2 \,, \tag{6.19}$$

where the beamforming vectors $\mathbf{v}_{\mathcal{R}_j^r(n)}$ and their power values ν_n can be found by maximizing the minimum of dual uplink SINR expressions:

$$\begin{aligned} \max_{\mathbf{v}_{\mathcal{R}_j^r(n)},\nu_n} \min_{n\in[t+\alpha]} \omega_n &= \frac{\nu_n |\mathbf{h}_{\mathbf{k}_j^r[n]}^H \mathbf{v}_{\mathcal{R}_j^r(n)}|^2}{\sum\limits_{b\,:\,\mathcal{R}_j^r[b]\ni \mathbf{k}_j^r[n]} \nu_b |\mathbf{h}_{\mathbf{k}_j^r[b]}^H \mathbf{v}_{\mathcal{R}_j^r(n)}|^2 + n_0} \\ s.t. \quad \sum_{n\in[t+\alpha]} \nu_n &\le P_T, \quad \|\mathbf{v}_{\mathcal{R}_j^r(n)}\|^2 = 1 \;\forall\; n \,. \end{aligned} \tag{6.20}$$

Note that the interference terms in the denominator of (6.20) have different indices compared with (6.18). The dual uplink optimization problem in (6.20) is quasi-convex and can be solved optimally for the given unicast group [82]. Here we use a standard iterative approach, where we adjust (e.g., by bisection) a target SINR value, denoted by $\bar{\omega}$, until the power constraint is met with a desired convergence level $|P_T - \sum_{n\in[t+\alpha]} \nu_n| < \epsilon$. However, this requires finding the power coefficients ν_n resulting in the minimum total power $\sum_{n\in[t+\alpha]} \nu_n$, for a given target SINR value $\bar{\omega}$. We use another internal iterative loop to address this issue. We first note that the (normalized) MMSE receiver $\mathbf{v}_{\mathcal{R}_j^r(n)} = \frac{\bar{\mathbf{v}}}{\|\bar{\mathbf{v}}\|}$, where $\bar{\mathbf{v}} = \frac{1}{\sqrt{\nu_n}} \Big(\sum_{b\,:\,\mathcal{R}_j^r[b]\ni \mathbf{k}_j^r[n]} \nu_b \; \mathbf{h}_{\mathbf{k}_j^r[b]} \mathbf{h}_{\mathbf{k}_j^r[b]}^H + n_0 \mathbf{I} \Big)^{-1} \mathbf{h}_{\mathbf{k}_j^r[n]}$ is the optimal RX beamformer solution for the dual uplink channel, for a fixed set of power values ν_n. Plugging this into (6.20), we can write the uplink SINR compactly as

$$\omega_n = \nu_n \mathbf{h}_{\mathbf{k}_j^r[n]}^H \Big(\sum_{b\,:\,\mathcal{R}_j^r[b]\ni \mathbf{k}_j^r[n]} \nu_b \; \mathbf{h}_{\mathbf{k}_j^r[b]} \mathbf{h}_{\mathbf{k}_j^r[b]}^H + n_0 \mathbf{I} \Big)^{-1} \mathbf{h}_{\mathbf{k}_j^r[n]} \,. \tag{6.21}$$

[12] We suspect that altering the placement scheme to remove this unevenness may improve the achievable rate due to the optimization problem's max-min structure.

Now, for a fixed $\bar{\omega}$, we can iteratively solve for optimal ν_n values resulting in the minimum total power, using the joint update method in [82]. This involves updating ν_n values via

$$\nu_n^{(m)} = \frac{\bar{\omega}}{\omega_n^{(m-1)}}\nu_n^{(m-1)} = \frac{\bar{\omega}}{\mathbf{h}_{\mathbf{k}_j^r[n]}^H\Big(\sum_{b:\mathcal{R}_j^r[b]\ni\mathbf{k}_j^r[n]}\nu_b^{(m-1)}\ \mathbf{h}_{\mathbf{k}_j^r[b]}\mathbf{h}_{\mathbf{k}_j^r[b]}^H + n_0\mathbf{I}\Big)^{-1}\mathbf{h}_{\mathbf{k}_j^r[n]}}\ , \tag{6.22}$$

until for every $n \in [t+\alpha]$, $\omega_n = \bar{\omega}$. Note that the convergence of the joint update method in (6.22) can be proved by the standard interference function approach [83].

So, in summary, we use an outer iterative loop to find the target SINR value $\bar{\omega}$ for which the power constraint is met, and an internal iterative loop to find the optimal power coefficients for any given $\bar{\omega}$. After the outer loop is converged, we calculate $\mathbf{v}_{\mathcal{R}_j^r(n)}$ and then we can find max-min optimal downlink beamformers using $\mathbf{w}_{\mathcal{R}_j^r(n)} = \rho_n\mathbf{v}_{\mathcal{R}_j^r(n)}$, where ρ_n is the downlink power associated with $\mathbf{w}_{\mathcal{R}_j^r(n)}$. To compute ρ_n, we first define $\mathbf{a} = [a_1\ a_2\ ...\ a_{t+\alpha}]$ and $\mathbf{D}$ to be a diagonal matrix of elements a_1, ..., $a_{t+\alpha}$, where

$$a_n = \frac{\omega_n}{(1+\omega_n)|\mathbf{h}_{\mathbf{k}_j^r(n)}^H\mathbf{v}_n|^2}\ . \tag{6.23}$$

Then, we define $\mathbf{G}$ as a matrix of elements $g_{n,b}$, $n, b \in [t+\alpha]$, where $g_{n,b} = |\mathbf{h}_{\mathcal{R}_j^r(n)}^H\mathbf{v}_b|^2$ if either $b = n$ or $\mathcal{R}_j^r[b] \ni \mathbf{k}_j^r[n]$, and $g_{n,b} = 0$ otherwise. Finally, defining $\boldsymbol{\rho} = [\rho_1\ \rho_2\ ...\ \rho_n]$, we have

$$\boldsymbol{\rho} = (\mathbf{I} - \mathbf{DG})^{-1}n_0\mathbf{a}\ . \tag{6.24}$$

6.3.6 Further Reduction in Subpacketization

Interestingly, with appropriate modifications, it is possible to further reduce the subpacketization requirement of cyclic caching by a factor of $(gcd(K,t,\alpha))^2$. This not only reduces the subpacketization requirement (and hence, the implementation complexity) considerably but also enables the subpacketization (and complexity) to be adjusted by tuning α (and K) parameter. For notation simplicity, let us simply use $\phi = gcd(K,t,\alpha)$ to represent the reduction factor. To achieve the reduced subpacketization, we use a user grouping technique inspired by [33]. The idea is to split users into groups of size ϕ and assume each group is equivalent to a *virtual* user. Then, we consider a virtual network consisting of these virtual users, in which the coded caching and spatial multiplexing gains are $\frac{t}{\phi}$ and $\frac{\alpha}{\phi}$, respectively. Finally, we apply the coded caching scheme proposed in Sections 6.3.1 and 6.3.2 to the virtual network, and *elevate* the resulting cache placement and delivery schemes to be applicable in the original network. The elevation procedure, which is thoroughly explained in Appendix C.2, is designed so that the maximum DoF of $t+\alpha$ is achieved without any increase in the required subpacketization. As a result, the elevated scheme would require the same subpacketization as the scheme applied to the virtual network, which is $\frac{K}{\phi}\big(\frac{t}{\phi}+\frac{\alpha}{\phi}\big)$, as there exist $\frac{K}{\phi}$ virtual users in the virtual network. Here, we explain the proposed procedure with the help of one example.

Example 7. *Assume a network scenario with $K = 8$, $t = 2$ and $\alpha = 4$. In this case, $\phi = 2$ and the resulting virtual network has $K' = \frac{K}{\phi} = 4$ virtual users, coded caching*

gain of $t' = \frac{t}{\phi} = 1$ *and spatial multiplexing gain of* $\alpha' = \frac{\alpha}{\phi} = 2$. *Assume the virtual users correspond to user groups* $v_1 = \{1,2\}$, $v_2 = \{3,4\}$, $v_3 = \{5,6\}$ *and* $v_4 = \{7,8\}$. *Applying the proposed cyclic caching scheme, the cache placement in the virtual network is* $\mathcal{Z}_{v_{k'}} = \{W^{(n)}_{k',q} \mid n \in [N], q \in [3]\}$, $\forall k' \in [4]$, *and the corresponding subpacketization requirement is* $K'(t'+\alpha') = 12$. *Data delivery is performed in four rounds, where three transmissions are done during each round. The first transmission vector at the first round is built as*

$$\mathbf{x}^{1'}_1 = \mathbf{w}'_{v_3} W^{(d_{v_1})}_{2,1} + \mathbf{w}'_{v_3} W^{(d_{v_2})}_{1,1} + \mathbf{w}'_{v_2} W^{(d_{v_3})}_{1,1} \,, \tag{6.25}$$

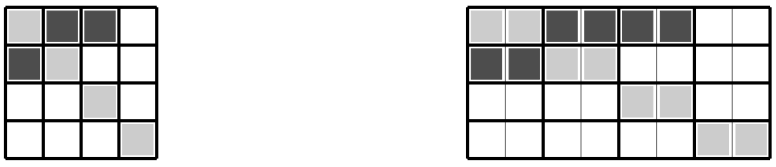
and other transmission vectors are also built similarly. Now, to elevate the cache placement to be applicable to the original network, we simply bind the cache content of each user in the original network with its corresponding virtual user in the virtual network. So, the cache placement for the original network is

$$\mathcal{Z}_1 = \mathcal{Z}_2 = \mathcal{Z}_{v_1}, \quad \mathcal{Z}_3 = \mathcal{Z}_4 = \mathcal{Z}_{v_2}, \quad \mathcal{Z}_5 = \mathcal{Z}_6 = \mathcal{Z}_{v_3}, \quad \mathcal{Z}_7 = \mathcal{Z}_8 = \mathcal{Z}_{v_4}.$$

Elevating the delivery procedure is more complex. Let us consider the first transmission of the first round, as provided in (6.25). *The first minifile in the transmission vector, i.e.,* $W^{(d_{v_1})}_{2,1}$, *is suppressed at user* v_3, *which will be equivalent to users* $\{5,6\}$ *in the original network. In the original network,* $W^{(d_{v_1})}_{2,1}$ *corresponds to two minifiles destined to users 1 and 2; i.e.,* $W^{(d_1)}_{2,1}$ *and* $W^{(d_2)}_{2,1}$. *Now, we see that the minifile* $W^{(d_1)}_{2,1}$, *in addition to being suppressed at the real users* $\{5,6\}$ *corresponding to virtual user* v_3, *it has also to be suppressed at real user* 2. *Similarly,* $W^{(d_2)}_{2,1}$ *has to be suppressed at real users* $\{5,6,1\}$. *Recalling that* $\alpha = 4$, *the suppression of the aforementioned minifiles at* 3 *users is possible. Analogous considerations can be done for the other two minifiles* $W^{(d_{v_2})}_{1,1}$ *and* $W^{(d_{v_3})}_{1,1}$ *of the first transmission for the virtual network. In this way, we see that the equivalent transmission vector to* (6.25) *for the original network will be*

$$\mathbf{x}^1_1 = \mathbf{w}_{5,6,2} W^{(d_1)}_{2,1} + \mathbf{w}_{5,6,1} W^{(d_2)}_{2,1} + \mathbf{w}_{5,6,4} W^{(d_3)}_{1,1} + \mathbf{w}_{5,6,3} W^{(d_4)}_{1,1} + \mathbf{w}_{3,4,6} W^{(d_5)}_{1,1} + \mathbf{w}_{3,4,5} W^{(d_6)}_{1,1} \,. \tag{6.26}$$

Other transmissions are built similarly. Overall, the algorithm requires subpacketization of 12 and delivers data in 12 transmissions. In comparison, applying cyclic caching without user grouping requires subpacketization of $K(t+L) = 48$ *(4 times higher than with user grouping), and the number of transmissions would be 48. A graphical representation for the first transmission of the first round, for both virtual and original networks, is provided in Figure 6.5.*



(a) Virtual Network (b) Original Network

Figure 6.5 – Graphical illustration of transmission vector in Example 7

6.4 Complexity and Performance Analysis

6.4.1 Complexity Analysis

For two main reasons, the subpacketization value is an important complexity indicator for any coded caching scheme. First, it indicates the number of smaller parts each file must be split into, for the scheme to work properly. As argued in [33], the exponentially growing subpacketization of conventional coded caching schemes can make their implementation infeasible, even for networks with a moderate number of users. Second, a scheme with smaller subpacketization *generally* requires a smaller number of transmissions, and consequently, fewer beamformer design problems to be solved. For comparison, in order to achieve the optimallinear one-shot sum DoF of $t+N_0$, cyclic caching requires subpacketization and transmission count of $\frac{K(t+\alpha)}{\phi^2_{K,t,N_0}}$ and $\frac{K(K-t)}{\phi^2_{K,t,N_0}}$, respectively. Both of these numbers are considerably smaller than in the original multi-antenna scheme of [8], for which the subpacketization is $\binom{K}{t}\binom{K-t-1}{N_0-1}$, while $\binom{K}{t+N_0}$ transmissions are needed in total. As each transmission requires solving a separate beamformer design problem, cyclic caching has remarkably lower computation complexity than in [47].

In addition to the performance and complexity benefits of reduced subpacketization, cyclic caching also has the critical advantage of relying on unicasting only, unlike other traditional schemes that rely on high-order multicast messages (e.g., using XOR-ed codewords). Although removing multicasting causes performance loss as discussed in [77], it enables high-performance optimized (MMSE-type) beamformers to be applicable with low complexity, even for large networks with a large sum-DoF. In the next subsection, we provide simulation results for large networks with up to $K = 100$ users, in which optimized beamformers are used. To the best of our knowledge, this is the first time a multi-antenna coded caching scheme has been applied with optimized beamformers to such a large network with a large sum-DoF.

From another perspective, cyclic caching also removes the requirement of decoding multiple data parts jointly at the same user during a single transmission. As a result, there is no need for complex receiver schemes such as successive interference cancellation (SIC). Cyclic caching requires that during all transmissions, every user in the target group decodes only one message, while in [8], $\binom{t+N_0-1}{t}$ terms must be jointly decoded. The receiver complexity reduction is possible also by splitting each transmission with overlapping multicast messages into multiple TDMA intervals as shown in [47], but it comes with a substantial subpacketization increase.

Interestingly, cyclic caching also enables tuning α and K parameters jointly to reduce both subpacketization and transmission count. This reduction is useful especially for large networks with a large number of users K, for which the complexity of coded caching schemes is critically limiting their practical implementation [33]. Selecting α to be smaller than the antenna count is straightforward and explained in [47]. However, in order to tune K, we consider a set of K_f additional *phantom* users and tune both α and K_f to maximize $gcd(K+K_f, t, \alpha)$. K_f phantom users are then omitted during all the subsequent transmissions. Of course, tuning either parameter comes with a DoF loss. For α, this is not necessarily an issue, especially when the communication is at finite-SNR.

Reduced subpacketization	Cyclic caching enables achieving the sum-DoF of $t+\alpha$, with the reduced subpacketization of $\frac{K(t+\alpha)}{\phi^2_{K,t,\alpha}}$.
Reduced number of transmissions	With cyclic caching, delivery to all users is completed with only $\frac{K(K-t)}{\phi^2_{K,t,\alpha}}$ transmissions.
Relying only on unicasting	Cyclic caching relies on unicasting only. As a result, optimized MMSE-type beamformers can be implemented with much lower complexity using the uplink-downlink duality. This improves the performance at low- and mid-SNR, compared with ZF beamforming.
No MAC decoding	Cyclic caching removes the requirement of MAC decoding, thus eliminating the necessity of complex receiver structures such as successive interference cancellation (SIC).
Controlling the complexity	The subpacketization of cyclic caching can be controlled by tuning the $\phi_{K,t,\alpha}$ parameter, which is possible by adjusting α and also by considering a set of K_f phantom users.

Table 6.2 – Advantages of the Cyclic Caching Scheme

In [47] it is shown that by choosing $\alpha < N_0$, one can obtain an improved beamforming gain, which considerably improves the performance at the finite-SNR regime. The joint impact of α and K_f tuning is studied through numerical simulations in Section 6.4.2.

In principle, the reduced subpacketization scheme of Section 6.3.6 can be applied by splitting users into groups of size $Q > 1$, such that $gcd(K, t, \alpha)$ is divisible by Q. This enables selecting several subpacketization levels, between $K(t+\alpha)$ and $\frac{K(t+\alpha)}{gcd(K,t,\alpha)^2}$. However, as we show later through simulations, the performance of cyclic caching is almost intact regardless of the selected subpacketization, and hence it makes sense to select the lowest possible subpacketization value.

In Table 6.2, we have summarized the key advantages of the cyclic caching scheme. Moreover, in Tables 6.3 and 6.4, we have compared the complexity order, in terms of both subpacketization requirement and the total number of transmissions, for the multi-antenna scheme of [8] (M-S), cyclic caching without user grouping (LIN), cyclic caching with user grouping (RED), the original group-based scheme in [33] (L-E), and the recently proposed scheme in [76] (M-B). In Table 6.3, the complexity order is provided for the case the global cache ratio $t = \frac{KM}{N}$ does *not* scale with the number of users K. For this case, we have simply used $\binom{K}{t} = \frac{K!}{t!(K-t)!} = O(K^t)$. However, if t scales with K (i.e., if $\gamma = \frac{M}{N}$ does not scale with K), this order approximation is no longer valid. In this case, to approximate $\binom{K}{t}$, we can consider the problem where we repeat a binary experiment with the success probability of γ for K times. As K grows large, according to the law of large numbers, the number of *typical* sequences (i.e., sequences with $K\gamma = t$ success outcomes) would be $\binom{K}{t}$. However, from information theory, we also know that the number of typical sequences approaches $H(\gamma)$, where $H(\gamma) = -\gamma \log_2 \gamma - (1-\gamma)\log_2(1-\gamma)$ is the entropy function. Hence, when t scales with K we can use the approximation $\binom{K}{t} = O(2^{KH(\gamma)})$, and the complexity order of different schemes would be as shown in Table 6.4. From

	M-S	LIN	RED	L-E	M-B
Subpacketization	$O(K^t K^{N_0-1})$	$O(K)$	$O(K)$	$O(K^{t/N_0})$	$O(K^t)$
Transmission Count	$O(K^{t+N_0})$	$O(K^2)$	$O(K^2)$	$O(K^{t/N_0+1})$	$O(K^{t+1})$

Table 6.3 – Complexity order of different schemes when t does not scale with K

	M-S	LIN	RED	L-E	M-B
Subpacketization	$O(2^{KH(\gamma)}K^{N_0-1})$	$O(K^2)$	$O(K^2)$	$O(2^{KH(\gamma)/N_0^2})$	$O(2^{KH(\gamma)})$
Transmission Count	$O(2^{KH(\gamma)})$	$O(K^2)$	$O(K^2)$	$O(2^{KH(\gamma)/N_0})$	$O(2^{KH(\gamma)})$

Table 6.4 – Complexity order of different schemes when t scales with K, $\gamma = \frac{M}{N}$, $H(\cdot)$ is the entropy function

Tables 6.3 and 6.4, it is clear that the complexity order of the proposed cyclic caching scheme is considerably smaller (linear if t does not scale with K, and quadratic otherwise) than all other schemes.

Finally, in Table 6.5, we have respectively compared the subpacketization requirement and the transmission count in some different network setups for M-S, LIN, RED, L-E, and M-B schemes. In the table, many entries are left empty for L-E and M-B schemes due to their tight restrictions on the network parameters. The L-E scheme is originally designed for networks in which $t \geq \alpha$ and α divides both t and K, while M-B requires $\frac{t+\alpha}{t+1}$ to be an integer.

From Table 6.5, it is clear that except for the L-E scheme, the RED scheme has the lowest subpacketization requirement among all the schemes. Also, regarding the number of transmissions, except for a specific case (the third row in the table) LIN and RED outperform all other schemes. It can be seen that, even when the user count is as low as $K = 30$, the M-S scheme becomes infeasible due to the substantial values for both the subpacketization and the transmission count. On the other hand, it is possible to implement RED even for a very large network of $K = 400$ users in which the spatial DoF is $\alpha = 100$. The results also show how proper tuning of α and K_f can help further reduce both the complexity and subpacketization in RED scheme.

6.4.2 Simulation Results

We use numerical simulations to compare the performance of cyclic caching with other schemes. The symmetric rate, as defined in (6.1), is used as the comparison metric. The L-E and M-B schemes are ignored in the simulations as they require tight restrictions on network parameters to work without significant performance (DoF) loss. Moreover, for the sake of comparison, we also consider a baseline scheme without coded caching, denoted by No-CC, in which only the local caching gain at each user is attained together with the spatial multiplexing gain. In the baseline scheme, we create K transmission vectors, where users $\{1, 2, ..., \alpha\}$ are served during the first transmission ($i = 1$), while for $i > 1$, the served user indices are a circular shift of the user indices targeted at transmission $i-1$. For all the simulations, we use maxmin-SINR optimal (MMSE-type) beamformers, found through solving (6.18). In the case of LIN, RED, and No-CC schemes, the beamformers

					Required Subpacketization				
K	t	N_0	α	K_f	M-S	LIN	RED	L-E	M-B
8	2	5	2	0	140	32	8	4	-
8	2	5	4	0	280	48	12	-	28
8	2	5	5	0	140	56	56	-	-
30	4	8	4	0	$> 10^7$	240	60	-	-
30	4	8	4	2	$> 10^8$	256	16	8	-
30	4	8	6	0	$> 10^9$	300	75	-	-
100	15	30	15	0	$> 10^{32}$	3000	120	-	-
100	15	30	15	5	$> 10^{33}$	3150	14	7	-
100	15	30	17	0	$> 10^{34}$	3200	3200	-	-
400	50	200	100	0	$> 10^{153}$	$> 10^4$	24	-	-
					Required number of transmissions (I)				
K	t	N_0	α	K_f	M-S	LIN	RED	L-E	M-B
8	2	5	2	0	70	48	12	6	-
8	2	5	4	0	28	48	12	-	28
8	2	5	5	0	8	48	48	-	-
30	4	8	4	0	$> 10^6$	780	195	-	-
30	4	8	4	2	$> 10^7$	832	52	28	-
30	4	8	6	0	$> 10^7$	780	195	-	$> 10^4$
100	15	30	15	0	$> 10^{32}$	8500	340	-	-
100	15	30	15	5	$> 10^{26}$	9450	42	21	-
100	15	30	17	0	$> 10^{26}$	8500	8500	-	$> 10^{17}$
400	50	200	100	0	$> 10^{113}$	$> 10^5$	56	-	-

Table 6.5 – Complexity comparison for some example network settings.

are designed with the uplink-downlink duality solution described in Section 6.3.5. For the M-S scheme, we use the more complex SCA method detailed in [47].

As discussed earlier, the complexity of the M-S scheme makes its implementation infeasible even for moderate-sized networks. In order to be able to compare all the schemes, we consider a small network of $K = 6$ users. The performance comparison results are provided in Figure 6.6. It can be seen that the performance values of LIN and RED schemes lie between M-S and No-CC. M-S provides better performance because it benefits from a *multicasting* gain; i.e., a single codeword (created with the XOR operation) benefits all the users in a multicast group. This multicasting effect is explained in more detail in [77], with the help of the so-called efficiency index parameter. On the other hand, No-CC scheme has the worst performance as it lacks the coded caching gain entirely. It should also be noted that choosing a smaller α value improves the performance of both M-S and LIN schemes at the lower SNR range, while this effect is more prominent for the LIN scheme.[13] This result is in line with the findings of [47, 48], where a smaller α is

[13]Note that when $\alpha = 1$, placement and delivery algorithms in M-S scheme are the same as the original scheme of [2]. The beamformer design also concedes with the max-min fair beamforming in [8].

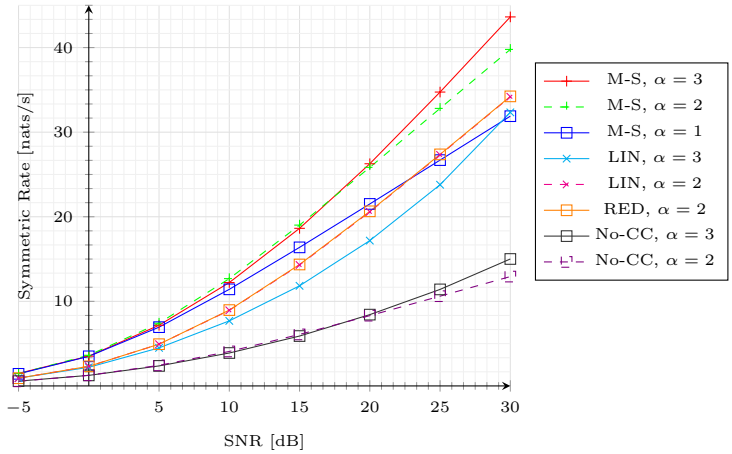
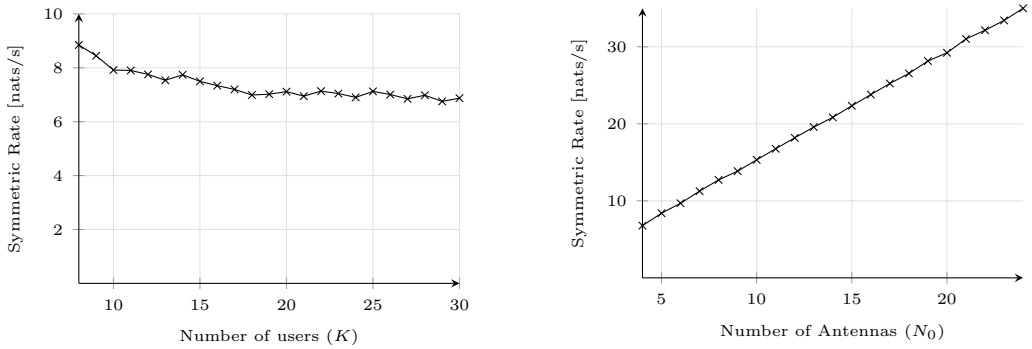


Figure 6.6 – M-S vs LIN vs RED vs No-CC rate; $K = 6$, $t = 2$, $N_0 = 3$



(a) Performance vs K, $N_0 = 4$

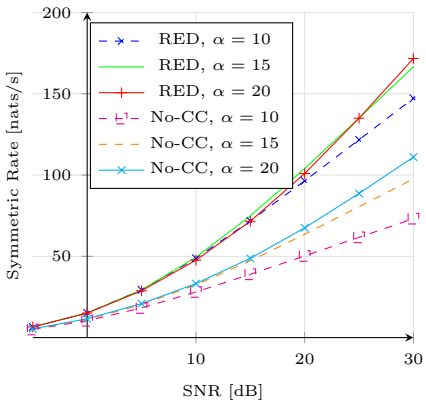
(b) Performance vs N_0, $K = 30$

Figure 6.7 – Performance of RED vs K and N_0 parameters, $t = 2$, $\alpha = N_0$, SNR= 10dB

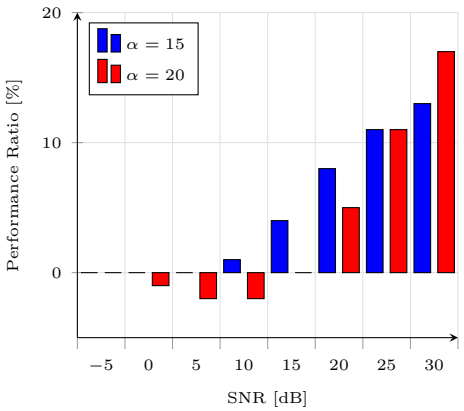
shown to improve the performance at the finite-SNR regime due to a better beamforming gain. Finally, Figure 6.6 demonstrates that RED scheme is able to provide the same performance as LIN scheme, but with lower complexity. This near-identical performance is a result of the two schemes having a very similar coding and interference-cancellation structure.

Unfortunately, although the M-S scheme has superior performance, its high complexity makes the implementation infeasible even for slightly larger networks. For example, for the small network of 6 users considered in Figure 6.6, with our simulation setup, the required time for simulating the M-S scheme is $O(10^3)$ times larger than the LIN scheme. As the network size grows, this ratio between the simulation times also grows exponentially. On the other hand, the simulation time for the RED scheme is roughly four times smaller than LIN. This is in line with the fact that for the considered network with $K = 6, t = 2, \alpha = 2$, the reduction in the subpacketization and transmission count is equal to $gcd(K, t, \alpha)^2 = 4$.

In Figure 6.7, we have analyzed the performance of RED scheme with respect to K

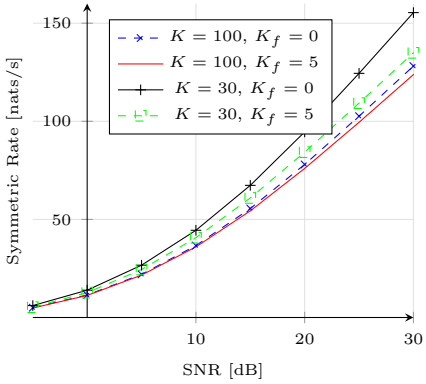


(a) Performance vs α

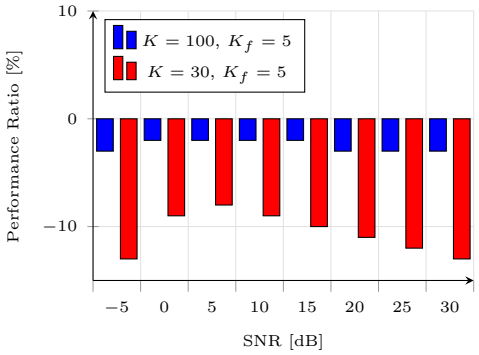


(b) Performance ratio w.r.t. $\alpha = 10$

Figure 6.8 – Performance of RED for a large network, $K = 100$, $t = 10$, $N_0 = 25$



(a) Performance vs K_f



(b) Performance ratio w.r.t. $K_f = 0$

Figure 6.9 – The effect of the K_f parameter, RED scheme, $t = 7$, $N_0 = 20$, $\alpha = 14$

and N_0 parameters, while assuming $t = 2$ and $\alpha = N_0$. From Figure 6.7a, we see that with the DoF value $t + N_0$ fixed, the performance slightly reduces as K increases. This is because we have assumed a fixed $t = \frac{KM}{N}$ value, which means the local caching gain $\frac{M}{N}$ is reduced as K is increased. On the other hand, from Figure 6.7b, we see that increasing L results in a linear increase in rate. This is simply because increasing N_0 improves DoF, and accordingly, the system performance.

Figure 6.8 illustrates the performance of RED scheme for a large network with $K = 100, t = 10$ and $N_0 = 25$. To the best of our knowledge, this is the first time a coded caching scheme is applied with optimized beamformers for such a large network. In Figure 6.8a, the results for No-CC scheme are included for comparison. It can be verified that decreasing α from 20 to 10 gives a small performance boost at the low- to moderate-SNR regime (< 15dB) due to improved beamforming gain. On the other hand, at larger SNR values, the $\alpha = 20$ setup performs much better due to the increased spatial multiplexing gain. These results are in line with the findings in [47].

Finally, the complexity reduction effect of the K_f parameter is analyzed in Figure 6.9

for two network scenarios of size $K_1 = 100$ and $K_2 = 30$ users. In both networks, we have assumed $t = 7$, $N_0 = 20$ and $\alpha = 14$. Without any phantom users, $gcd(K, t, \alpha) = 1$ and the subpacketization values for the two networks are $S_1 = 2100$ and $S_2 = 630$, respectively. However, by adding only $K_f = 5$ phantom users, the common denominator becomes $gcd(K + K_f, t, \alpha) = 7$, and hence, the subpacketization values are reduced to $S_1 = 45$ and $S_2 = 15$, respectively. Interestingly, the decrease in the achievable rate, due to adding K_f phantom users, is relatively minor for both networks. In fact, for the larger network, the performance loss is less than 4% over the entire SNR range, while for the smaller network, the deterioration is less than 15%. On the other hand, adding the extra phantom users reduces the simulation time in our setup by a factor of ~ 10.

Chapter 7

Coded Caching with Heterogeneous Quality-of-Service Requirements

In this chapter we focus on the cache-aided K-user shared-link, where each user's request comes with a certain *Quality-of-Service* (*QoS*) requirement, thus allowing – in the context of multi-layered coding – users to download only those file layers that are necessary to meet their own *QoS* requirements. For such a problem we propose a generic *QoS*-aware delivery scheme that works for a class of uncoded caching schemes within which we identify a specific cache placement that is proven to be optimal if during the caching phase the server knows only the number of the users requiring a given *QoS* but not their identity. Finally, we will also touch upon the case when, in the caching phase, the server does not have any information about the users' *QoS* requirements.

7.1 Context and Related Works

Most streaming services employ adaptive streaming techniques to serve videos with different quality levels, often as a function of the available bandwidth, processing power, type of subscription etc. Indeed in current video coding standards such us H.264/AVC, this adaptability is a main ingredient, and the different quality levels are obtained via scalable video coding which encodes videos into many streams/layers such that the more streams the user decodes, the higher is the video quality. This direction of coded caching with heterogeneous quality-of-service requirements, was first studied in [84] by Yang and Gündüz, who presented — for the case where the users' *QoS* requirements are known during the cache-placement phase — two coded caching schemes and a cut-set type converse. Subsequent similar work in [85] proposes — for the case of the cache-aided gaussian broadcast channel — different schemes that exploit multi-layer source coding to serve at higher rates those users with better channels. Another interesting work that merges the benefits of coded caching and multi-layered source coding is presented in [86] for the setting of 2-layered files, where different transmitting nodes serve K cache-enabled users who can decode the base layer and, if their channel strength allows, the enhancement layer as well. Other works that jointly study coded caching and multi-layered coded files

can also be found in [87, 88].

Caching oblivious to specific QoS requirements Unlike other related works though, our work focuses on the case where, at the time of cache placement, the server knows only many users want a certain *QoS* level, but it does not know which *QoS* level each particular user wants. Having caching that is oblivious to the specific *QoS* requirement of each user, can be of particular interest because the cache placement can indeed take place long before the allowed or desired *QoS* is established.

7.2 System Model and Problem Definition

As in the original MAN setting, we consider the K-user cache-aided shared-link, where a transmitter with access to a library of N files $W^{(1)}, W^{(2)}, \dots, W^{(N)}$, is tasked with serving a set $\{1, 2, \dots, K\}$ of K users equipped with a cache of size M ($\gamma = \frac{M}{N}$). Using successive refinement source coding, each library file $W^{(n)},\ n \in [N]$ is split into $\mathcal{H}$ layers as follows $W^{(n)} = \{W^{(n,1)}, W^{(n,2)}, \dots, W^{(n,\mathcal{H})}\}$, where $W^{(n,h)},\ h \in [\mathcal{H}]$ denotes the $h-$th layer of the $n-$th file. As usual we assume that the system operates in two phases: the cache placement phase and the delivery phase. In the first phase, the users fill their cache with part of the content of the library and we denote Z_k the cache content by users k such that the vector $\mathbf{Z} = (\mathcal{Z}_1, \mathcal{Z}_2, \dots, \mathcal{Z}_K)$ denotes the overall cache placement strategy. In the delivery phase, we assume that when a user k requests a file $W^{(d_k)}, d_k \in [N]$, it has a given *QoS* requirement $h \in [\mathcal{H}]$, meaning that this user must be succesfully delivered the first h layers $\{W^{(d_k,1)}, W^{(d_k,2)}, \dots, W^{(d_k,h)}\}$ of its desired file $W^{(d_k)}$. Assuming that each file n has normalized unit size $|W^{(n)}| = 1$, we will use r_h to denote the size of these desired h layers, i.e. $r_h \triangleq \sum_{j=1}^{h} |W^{(n,j)}|$. Naturally $\sum_{l=1}^{\mathcal{H}} |W^{(n,l)}| = r_{\mathcal{H}} = |W^{(n)}| = 1,\ \forall n \in [N]$. We define *user-QoS association* the partition of the set of users $[K]$ as

$$\mathcal{U} = \{\mathcal{U}_1, \mathcal{U}_2, .., \mathcal{U}_{\mathcal{H}}\} \tag{7.1}$$

telling us exactly the *QoS* level of each user, where users in set $\mathcal{U}_h$ must be delivered exactly (and only) layers $1, 2, \dots, h$ of their respective requested file. Related to this, the number $K_h = |\mathcal{U}_h|$ denotes the number of users with *QoS* level $h \in [\mathcal{H}]$, and we define *QoS profile*

$$\mathbf{K} = (K_1, K_2, \dots, K_{\mathcal{H}}).$$

Each *QoS profile* $\mathbf{K}$ defines a class $\mathcal{U}_{\mathbf{K}}$ comprising of all $\mathcal{U}$ that share the same profile[1] $\mathbf{K}$.

Key assumptions and objective As usual, we will assume that the cache placement phase is oblivious to the subsequent demand vector $\mathbf{d} = (d_1, d_2, \dots, d_K)$. In addition, here, we will assume that the cache placement phase is also oblivious to the user-*QoS* association $\{\mathcal{U}_1, \mathcal{U}_2, .., \mathcal{U}_{\mathcal{H}}\}$ but that it is aware of the *QoS* profile $\mathbf{K}$. When the delivery

[1] We can easily see that the number of different partitions $\mathcal{U}$ associated to any fixed $\mathbf{K}$, is given by the well known multinomial coefficient $\binom{K}{K_1, \cdots, K_{\mathcal{H}}}$.

phase starts the serve is notified of the demand vector $\mathbf{d}$ and of the exact user-*QoS* association $\{\mathcal{U}_1,\mathcal{U}_2,..,\mathcal{U}_{\mathcal{H}}\}$.

For any tuple $(\mathcal{U},\mathbf{d},\mathbf{Z})$, we define $T^*(\mathcal{U},\mathbf{d},\mathbf{Z})$ as the optimal delivery time necessary to serve all users' requests. Under the assumption that the exact user-*QoS* association is not known during the cache placement, which is only aware of the *QoS* profile $\mathbf{K}$, the optimal worst-case delivery time can be formally define as

$$T^*(\mathbf{K}) \triangleq \min_{\mathbf{Z}} \max_{(\mathcal{U},\mathbf{d})\in(\mathcal{U}_{\mathbf{K}},[N]^K)} T^*(\mathcal{U},\mathbf{d},\mathbf{Z}), \tag{7.2}$$

for any given *QoS* profile $\mathbf{K}$. As in the previous problems discussed in this thesis, the minimization is over all *uncoded* cache placement strategies.

7.3 Main Results

We proceed with the main result that identifies the optimal performance $T^*(\mathbf{K})$.

Theorem 10. *In the K-user cache-aided shared-link network with $\mathcal{H}$-layer file coding, where the cache placement phase is aware of the* QoS *profile $\mathbf{K}$ but it is unaware of the user-*QoS *association $\mathcal{U}$, the optimal worst-case delivery time takes the form*

$$T^*(\mathbf{K}) = \sum_{h=1}^{\mathcal{H}} \sum_{g=0}^{K} \underbrace{\frac{\sum_{r=1}^{\mathfrak{P}_{g,h}} \binom{K-r}{g}}{\binom{K}{g} N}}_{c_{g,l}} x^*_{g,h} \tag{7.3}$$

where $\mathfrak{P}_{g,h} = \min\left\{K-g, K-\sum_{j=1}^{h-1} K_j\right\}$ and where the set $\{x^*_{g,h}\}_{h\in[\mathcal{H}],g\in[K]_0}$ is the optimal point of the linear program

$$\underset{x_{g,h}}{\text{minimize}} \quad \sum_{h=1}^{\mathcal{H}} \sum_{g=0}^{K} c_{g,h} x_{g,h} \tag{7.4}$$

$$\text{subject to} \quad \sum_{g=0}^{K} x_{g,h} = (r_h - r_{h-1})N, \quad h = 1,\dots,\mathcal{H} \tag{7.5}$$

$$\sum_{g=0}^{K} g \cdot \left(\sum_{h=1}^{\mathcal{H}} x_{g,h}\right) \leq KM, \tag{7.6}$$

$$x_{g,h} \geq 0, \quad h = 1,\dots,\mathcal{H} \quad g = 0,1,\dots,K. \tag{7.7}$$

where $\{x_{g,h}\}_{h\in[\mathcal{H}],g\in[K]_0}$ are the optimization variables.

The performance of Theorem 10 is achieved by the scheme presented in Section 7.4. The proposed scheme works for any set $\{x_{g,h}\}_{h\in[\mathcal{H}],g\in[K]_0}$ satisfying equations (7.5),(7.6),(7.7), but the optimal performance in (7.3) is achievable by using $\{x^*_{g,h}\}_{h\in[\mathcal{H}],g\in[K]_0}$. The matching converse — as we will see in Section 7.5 — coincides with the optimal

value of the linear program (LP) in (7.4)-(7.7) which is directly used to design the scheme that optimally reflects the *QoS* profile $\mathbf{K}$.

A peculiar characteristic of the above result is the fact that the achievable optimal performance — under the considered assumptions — is obtained by leveraging the constructed lower bound. In fact, the lower bound tells us exactly — for each file — how much of each layer we have to cache in each user' cache, thus highlighting the importance of developing tight bounds.

We now proceed by providing an achievable performance for the case where caching is oblivious to $\mathbf{K}$.

Proposition 4. *In the K-user cache-aided shared-link network with $\mathcal{H}$-layer file coding, and with caching that is oblivious to any* QoS *information about the users, the following delivery time is achievable*

$$T_{obl}(\mathbf{K}) = \sum_{h=1}^{\mathcal{H}} \frac{\sum_{r=1}^{\mathfrak{P}_{g,h}} \binom{K-r}{K\gamma}}{\binom{K}{K\gamma}} (r_h - r_{h-1}), \tag{7.8}$$

where $\mathfrak{P}_{K\gamma,h} = \min\left\{K - K\gamma, K - \sum_{j=1}^{h-1} K_j\right\}$.

The achievable scheme stated in the above proposition, is a by-product of the general scheme presented in Section 7.4 and it is obtained by essentially properly choosing feasible values $x_{g,l}$ that do not depend on the *QoS* profile $\mathbf{K}$. This 'oblivious' scheme achieving (7.8), is compared in the plot in Fig. 7.1, to the optimal scheme that employs the knowledge of $\mathbf{K}$ at the cache placement scheme.

7.4 Achievable Caching and Delivery Scheme

In this section we present the general caching and delivery schemes that achieve the optimal delay of Theorem 10 as well as the achievable performance in Proposition 7.8.

7.4.1 Cache Placement Scheme

For any $n \in [N]$ and any $h \in [\mathcal{H}]$, the first step is to split each subfile $W^{(n,h)}$ into $K+1$ mini-files $W^{(n,h)} = \{W^{(n,h,g)}\}_{g=0}^{K}$. We denote the size of each such mini-file as $y_{g,h}$, i.e. $y_{g,h} \triangleq |W^{(n,h,g)}|$ for any $n \in [N]$. Because of the layer coding technique and the limited memory of each user cache, the following equations must be satisfied

$$\sum_{g=0}^{K} y_{g,h} = (r_h - r_{h-1}), \quad h = 1, \dots, \mathcal{H} \tag{7.9}$$

$$\sum_{g=0}^{K} g \cdot \left(\sum_{h=1}^{\mathcal{H}} y_{g,h}\right) \leq K\gamma, \tag{7.10}$$

$$y_{g,h} \geq 0, \quad h = 1, \dots, \mathcal{H} \quad g = 0, 1, \dots, K, \tag{7.11}$$

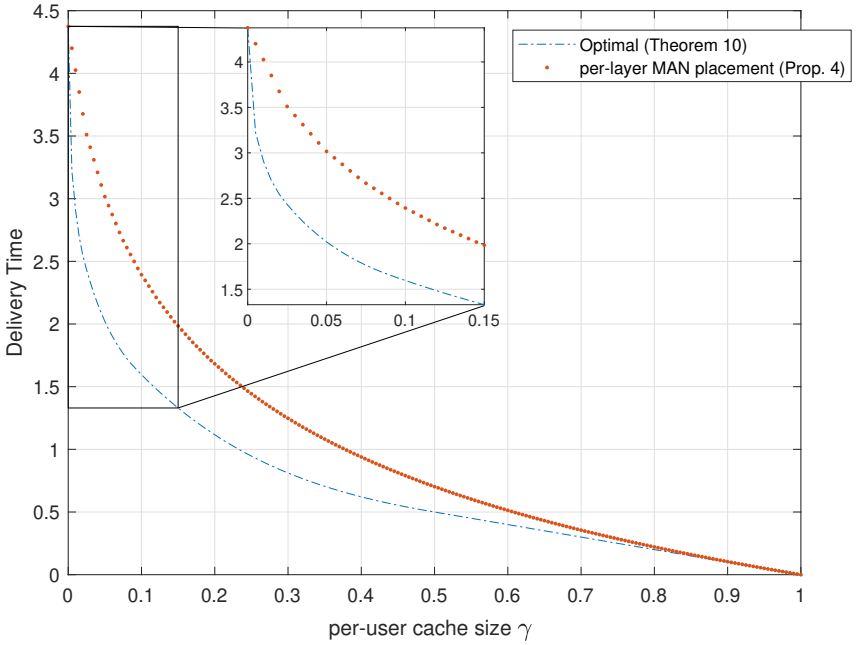


Figure 7.1 – Comparison of delivery time between the optimal scheme and the 'oblivious' one. This is done for *QoS* profile $\mathbf{K} = \{12, 10, 8, 5, 3, 2, 1\}$ and sizes $r_h = 2^{h-\mathcal{H}}, \forall h \in [\mathcal{H}], \mathcal{H} = 7$.

which correspond to the one in equations (7.5)-(7.7) after defining $x_{g,h} \triangleq y_{g,h}N$ for any $g \in [K]_0$ and any $h \in [H]$.

Next, we further split each mini-file $W^{(n,h,g)}$ into $\binom{K}{g}$ equally-sized micro-files $W_\tau^{(n,h,g)}$ as follows $W^{(n,h,g)} = \{W_\tau^{(n,h,g)} | \ |\tau| = g, \tau \subset [K]\}$.

At this point we can fill the cache $\mathcal{Z}_k$ of each user k, by placing in it, from each file $W^{(n)}$, $n \in [N]$, all the micro-files $W_\tau^{(n,h,g)}$ for any $\tau \ni k$, as described below

$$\mathcal{Z}_k = \{W_\tau^{(n,h,g)} \mid k \in \tau, n \in [N], h \in [\mathcal{H}], g \in [K]_0\}.$$

We now prove that the above placement adheres to the cache-size constraint.

Compliance of the placement scheme with the cache constraint

We first recall that for each mini-file $W^{(n,h,g)}, n \in [N], h \in [\mathcal{H}], g \in [K]_0$, any derived micro-file $W_\tau^{(n,h,g)}$ with $|\tau| = g, \tau \subset [K]$, have size $|W_\tau^{(n,h,g)}| = \frac{y_{g,h}}{\binom{K}{g}}$. Due to the symmetry in the cache placement, each user $k \in [K]$ stores exactly $\binom{K-1}{g-1}$ micro-files $W_\tau^{(n,h,g)}$ from any fixed mini-file $W^{(n,h,g)}$, which means that, for each mini-file $W^{n,h,g}$, each user stores $\binom{K-1}{g-1}\frac{y_{g,h}}{\binom{K}{g}} = \frac{g}{K}y_{g,h}$ units of file. Consequently the total amount of data from the $h-$th layer of file $W^{(n)}, n \in [N]$ stored by each user k, takes the form $S_k(W^{(n,h)}) \triangleq \sum_{g=0}^{K} \frac{g}{K}y_{g,h}$,

and thus the total memory employed by each user to cache a portion of a file $W^{(n)}, n \in [N]$, takes the form

$$S_k(W^n) = \sum_{h=1}^{\mathcal{H}} S_k(W^{(n,h)}) = \sum_{h=1}^{\mathcal{H}} \sum_{g=0}^{K} \frac{g}{K} y_{g,h} = \frac{1}{KN} \sum_{g=0}^{K} g \sum_{h=1}^{\mathcal{H}} x_{g,h}.$$

Finally, summing over all library files, we see that each user caches

$$\sum_{n=1}^{N} \frac{1}{KN} \sum_{g=0}^{K} g \sum_{h=1}^{\mathcal{H}} x_{g,h} = \frac{1}{K} \sum_{g=0}^{K} g \sum_{h=1}^{\mathcal{H}} x_{g,h} \text{ (units of file)}$$

which does not exceed M due to the constraint in (7.6). □

Cache placement allowing the achievable performances in Theorem 10 and in Proposition 7.8 The cache placement that allow to achieve the optimal performance stated in equation 7.3 is obtained from the strategy described above by selecting $x_{g,h} = x^*_{g,h}$, where $\{x^*_{g,h}\}_{h\in[\mathcal{H}],g\in[K]_0}$ is the solution of the linear program in (7.4)-(7.7). On the other hand, the performance in Proposition 7.8 is obtained by selecting, for any $h \in [\mathcal{H}]$, $x_{g,h}$ as

$$x_{g,h} \begin{cases} 0 & g \neq K\gamma \\ N(r_h - r_{h-1}) & g = K\gamma, \end{cases} \tag{7.12}$$

which we can see does not depend on $\mathbf{K}$.

7.4.2 Delivery Scheme

The delivery phase starts when the users reveal their requested file and their *QoS* requirement to the server which uses the delivery scheme described below to serve all the requests. The scheme sequentially serves all the requested files, and does so layer by layer, in accordance to the now known user-*QoS* association $\{\mathcal{U}_1, \mathcal{U}_2, .., \mathcal{U}_{\mathcal{H}}\}$. The scheme operates in $\mathcal{H}$ rounds, one for each layer, where in particular, round $h \in [\mathcal{H}]$ serves the $h-$th layer of the files requested by the users in the set $\mathfrak{G}_h \triangleq \bigcup_{j=h}^{\mathcal{H}} \mathcal{U}_j$, which have a *QoS* level of h or above. Specifically, round h aims to deliver the missing parts of subfiles $W^{(d_k,h)}, \forall k \in \mathfrak{G}_h$.

Each round h is split into Q_h sub-rounds, where $Q_h \leq K+1$ is the total number of non-zero elements in the aforementioned $\{x^*_{g,h}\}_{h\in[\mathcal{H}],g\in[K]_0}$. Consequently each sub-round $g \in [Q_h]$ serves subfiles $W^{(d_k,h,g)}$ for all $k \in \bigcup_{j=h}^{\mathcal{H}} \mathcal{U}_j$.

In each such sub-round g of round h, we apply a variation of the delivery scheme in [2], where we create $\binom{K}{g+1}$ sets $\mathcal{Q} \subseteq [K]$ of size $|\mathcal{Q}| = g+1$, and for each such set $\mathcal{Q}$, we pick the set of users $\chi_{\mathcal{Q}} = \mathcal{Q} \cap \mathfrak{G}_h$. If $\chi_{\mathcal{Q}} \neq \emptyset$, the server transmits

$$X_{\chi_{\mathcal{Q}}} = \bigoplus_{k\in\chi_{\mathcal{Q}}} W^{(d_k,h,g)}_{\mathcal{Q}\setminus\{k\}} \tag{7.13}$$

else if $\chi_{\mathcal{Q}} = \emptyset$, there is no transmission, and we move to the next $\mathcal{Q}$.

The decoding follows directly from [2].

Calculation of Delivery Time

We first recall that $|W^{(d_k,h)}| = r_h - r_{h-1}$ and that $|W^{(d_k,h,g)}| = \frac{x_{g,h}}{N}$. We also note that in each sub-round g of round l, the total number of transmissions is

$$G_{h,g} = \binom{K}{g+1} - \binom{\sum_{j=1}^{h-1} K_j}{g+1}$$

where the second term accounts for the number of times the set $\chi_{\mathcal{Q}}$ was empty. It is easy to see that if $\sum_{j=1}^{h-1} K_j < g+1$, we have $G_{h,g} = \sum_{r=1}^{K-g} \binom{K-r}{g}$, else we have

$$\begin{aligned} G_{h,g} &= \binom{K-1}{g} + \binom{K-2}{g} + \cdots + \binom{\sum_{j=1}^{h-1} K_j}{g} \\ &= \sum_{r=1}^{K-\sum_{j=1}^{h-1} K_j} \binom{K-r}{g} \end{aligned}$$

which implies that

$$G_{h,g} = \sum_{r=1}^{\min\{K-g, K-\sum_{j=1}^{h-1} K_j\}} \binom{K-r}{g}.$$

Each transmission has normalized duration $\frac{x_{g,l}}{N\binom{K}{g}}$, and thus the total duration of a sub-round is

$$\frac{\sum_{r=1}^{\min\{K-g, K-\sum_{j=1}^{h-1} K_j\}} \binom{K-r}{g}}{\binom{K}{g} N} x_{g,h}.$$

Summing over all sub-rounds and then over all rounds, we calculate the total delivery time to be

$$T^*(\mathbf{K}) = \sum_{h=1}^{\mathcal{H}} \sum_{g=0}^{K} \frac{\sum_{r=1}^{\min\{K-g, K-\sum_{j=1}^{h-1} K_j\}} \binom{K-r}{g}}{\binom{K}{g} N} x_{g,l}. \tag{7.14}$$

It is easy to verify that as a by-product we get the achievable delivery times stated in Thereom 10 and Proposition 7.8.

7.5 Information Theoretic Converse

In this section we present the information theoretic converse that proves, in conjunction with the achievable scheme in Section 7.4, the optimality of the delivery time in Theorem 10. The proof of converse draws nicely again from the technique in [52] (and its adaptation in Chapter 3), that translates the coded caching problem into an equivalent index coding problem, making use of the cut-set type outer bound on the index coding capacity introduced in [89, Corollary 1].

The coded caching problem here is uniquely determined when the users' requests and *QoS* levels $(\mathbf{d},\mathcal{U})$ are revealed to the server. Focusing on the worst-case scenario with $N \geq K$, we define

$$\mathcal{D}_{\mathbf{K}} \triangleq \{\mathbf{d}(\mathcal{U}) : \boldsymbol{d} \in \mathcal{D}_{wc}, \mathcal{U} \in \mathcal{U}_{\mathbf{K}}\}$$

as the set of worst-case demands associated to a given *QoS profile* $\mathbf{K}$, where $\mathcal{D}_{wc}$ is the set of all demand vectors $\mathbf{d}$ that are comprised of (the indices of) K different files. First of all, we proceed to translate the delivery phase of the considered problem into an index coding problem, similarly to as we did in Section 3.4.

Translation to index coding A first step in converting the caching problem defined by $(\mathbf{d},\mathcal{U})$ into an equivalent index coding problem, is to split a requested layer of a requested file in the most general way. In particular, focusing on layer h, this means that we split any requested subfile $W^{(d_i,h)},\ i \in \cup_{p=h}^{\mathcal{H}}\mathcal{U}_p$ into 2^K disjoint subfiles $W_{\tau}^{(d_i,h)}, \tau \in 2^{[K]}$, where we recall that $2^{[K]}$ is the power set of $[K]$, and where $\tau \in [K]$ indicates the set of users in which $W_{\tau}^{(d_i,h)}$ is cached. In the context of index coding, we can view each such subfile $W_{\tau}^{(d_i,h)}$ as a message requested by a different user that has as side information all the content in the cache of the requesting user from the caching problem. Given this aforementioned representation of the requested files, the corresponding index coding problem is fully defined by the side information graph $\mathcal{G}_{\mathcal{U},\mathbf{d}} = (\mathcal{V}_{\mathcal{G}}, \mathcal{E}_{\mathcal{G}})$, where $\mathcal{V}_{\mathcal{G}}$ (which has cardinality $|\mathcal{V}_{\mathcal{G}}| = 2^{K-1} \cdot \sum_{j=1}^{\mathcal{H}} jK_j$) is the set of vertices corresponding to the requested subfiles $W_{\tau}^{(d_i,h)}$, and where $\mathcal{E}_{\mathcal{G}}$ is the set of directed edges of the graph. A directed edge from vertex $v \in \mathcal{V}_{\mathcal{G}}$ to $v' \in \mathcal{V}_{\mathcal{G}}$ exists if and only if the index coding user requesting the subfile corresponding to vertex v', knows the subfile corresponding to vertex v.

Now that we have converted the coded caching problem into an index coding problem we make use again of the so-called MAIS bound already stated in Lemma 1, which we use it here with $N_0 = 1$ since in this chapter we are considering a single antenna transmitter. Lemma 1 will be used to lower bound the delay $T^*(\mathcal{U},\mathbf{d},\mathbf{Z})$. To this end, we proceed to carefully select acyclic subgraphs that will yield, via Lemma 1, tighter lower bounds for $T^*(\mathcal{U},\mathbf{d},\mathbf{Z})$. In the following lemma we use σ to denote a permutation on the tuple $(1,2,\ldots,K)$.

Lemma 8. *All subfiles*

$$W_{\tau_i}^{(d_{\sigma(i)},h)},\ \forall i \in [K],\ \forall h : \sigma(i) \in \bigcup_{p=h}^{\mathcal{H}} \mathcal{U}_p,\ \forall \tau_i \subseteq [K] \setminus \{\sigma(1),\ldots,\sigma(i)\} \tag{7.15}$$

such that

$$\sigma(i) \in \mathcal{U}_p\ \wedge\ \sigma(j) \in \mathcal{U}_w \text{ with } i \leq j \iff p \geq w \tag{7.16}$$

compose an acyclic subgraph $\mathcal{J}$ of $\mathcal{G}_{\mathcal{U},\mathbf{d}}$.

We notice that, for any permutation σ that does not necessarily satisfies (7.16), the fact that the subfiles in (7.15) form an acyclic subgraph of the corresponding side information graph is a simple direct extension of Lemma 1 in [52] to the considered setting with multi-layered coded files. Among all the permutations[2] $\sigma \in S_K$, the permutations that satisfy 7.16 are those that induce acyclic subgraphs with the highest number of nodes. This latter fact is crucial for the derivation of a tight converse bound. The total number of permutations σ that satisfy the condition in (7.16) can be easily calculated to be $K_1!K_2!\cdots K_{\mathcal{H}}!$. We will denote the set of all such permutations by $\mathfrak{R}$.

After choosing an acyclic subgraph according to Lemma 8, we return to Lemma 1 and form the following lower bound

$$T^*(\mathcal{U},\mathbf{d},\mathbf{Z}) \geq T^{lb}_{\sigma}(\mathcal{U},\mathbf{d},\mathbf{Z}) \tag{7.17}$$

where

$$\begin{aligned} T^{lb}_{\sigma}(\mathcal{U},\mathbf{d},\mathbf{Z}) \triangleq & \sum_{h=1}^{\mathcal{H}}\sum_{j=1}^{K_{\mathcal{H}}} \sum_{\tau_j\subseteq[K]\setminus\{\sigma(1),\cdots,\sigma(j)\}} |W^{(d_{\sigma(j)},h)}_{\tau_j}| + \sum_{h=1}^{\mathcal{H}-1}\sum_{j=1}^{K_{\mathcal{H}-1}} \sum_{\tau_j\subseteq[K]\setminus\{\sigma(1),\cdots,\sigma(K_{\mathcal{H}}+j)\}} |W^{(d_{\sigma(K_{\mathcal{H}}+j)},h)}_{\tau_j}| \\ & + \cdots + \sum_{h=1}^{1}\sum_{j=1}^{K_1} \sum_{\tau_j\subseteq[K]\setminus\{\sigma(1),\ldots,\sigma(K-K_1+j)\}} |W^{(d_{\sigma(K-K_1+j)},h)}_{\tau_j}|. \end{aligned} \tag{7.18}$$

Since (7.18) holds for each $\sigma \in \mathfrak{R}$, we proceed to lower bound $T^*(\mathcal{U},\mathbf{d},\mathbf{Z})$ with the following average

$$T^*(\mathcal{U},\mathbf{d},\mathbf{Z}) \geq \frac{1}{|\mathfrak{R}|}\sum_{\sigma\in\mathfrak{R}} T^{lb}_{\sigma}(\mathcal{U},\mathbf{d},\mathbf{Z}). \tag{7.19}$$

We now recall that our interest lies on the worst-case delay scenario for a given profile $\mathbf{K}$. Relaxing the maximization with an average we get

$$T^*(\mathbf{K}) = \min_{\mathbf{Z}} \max_{(\mathcal{U},\mathbf{d})\in(\mathcal{U}_{\mathbf{K}},[N]^K)} T^*(\mathcal{U},\mathbf{d},\mathbf{Z}) \tag{7.20}$$

$$\geq \min_{\mathbf{Z}} \frac{1}{|\mathcal{U}_{\mathbf{K}}|\cdot|\mathcal{D}_{wc}|} \sum_{\mathcal{U}\in\mathcal{U}_{\mathbf{K}}}\sum_{\mathbf{d}\in\mathcal{D}_{wc}} T^*(\mathcal{U},\mathbf{d},\mathbf{Z}), \tag{7.21}$$

$$\geq \min_{\mathbf{Z}} \frac{1}{|\mathcal{U}_{\mathbf{K}}|\cdot|\mathcal{D}_{wc}|\cdot|\mathfrak{R}|} \sum_{\mathcal{U}\in\mathcal{U}_{\mathbf{K}}}\sum_{\mathbf{d}\in\mathcal{D}_{wc}}\sum_{\sigma\in\mathfrak{R}} T^{lb}_{\sigma}(\mathcal{U},\mathbf{d},\mathbf{Z}), \tag{7.22}$$

where $\mathcal{D}_{wc}$ is the set of all demands $\mathbf{d}$ that are comprised of the indices of K different files and where $T^{lb}_{\sigma}(\mathcal{U},\mathbf{d},\mathbf{Z})$ is given by (7.18).

We can now rewrite the inequality in (7.22) in the form

$$T^*(\mathbf{K}) \geq \min_{\mathbf{Z}} \sum_{h=1}^{\mathcal{H}}\sum_{g=0}^{K} \sum_{\tau\subset[K]:|\tau|=g}\sum_{n=1}^{N} c^{(n,h)}_{\tau}|W^{(n,h)}_{\tau}|, \tag{7.23}$$

[2]We recall that S_K is the symmetric group of all k-permutatios f the set $[K]$.

where $c_\tau^{(n,h)}$ is evaluated in what follows. To this end, we first notice that the averages in (7.22) act as a symmetrization step such that all subfiles that are cached in exactly g users, i.e. those with lower index τ such that $|\tau| = g$, appear an equal number of times in the summation in (7.22), thus implying that $c_\tau^{(n,h)} = c_{\tau'}^{(n,h)}$ for $|\tau| = |\tau'| = g$ for any $n \in [N]$ and any $h \in [\mathcal{H}]$. At the same time the fact that we average over all possible demand vectors with distinct requested files and over all possible QoS-user associations satisfying the QoS profile $\mathbf{K}$, we have that the coefficients $c_\tau^{(n,h)}$ do not depend on the specific n. This implies that

$$c_\tau^{(n,h)} = c_{\tau'}^{(n',h)} \ \forall \tau, \tau' : |\tau| = |\tau'| = g \text{ and } \forall n, n' \in [N]. \tag{7.24}$$

Thus, the expression in (7.23) can be simplified to

$$T^*(\mathbf{K}) \geq \min_{\mathbf{Z}} \sum_{h=1}^{\mathcal{H}} \sum_{g=0}^{K} c_{g,h} x_{g,h}, \tag{7.25}$$

where $x_{g,h} \triangleq \sum_{n\in[N]} \sum_{\mathcal{T}\subseteq[K]:|\tau|=g} |W_\tau^{(n,h)}|$.

Because of the symmetry created, we can now calculate $c_{g,h}$ directly from (7.18) as the percentage of subfiles of layer h with $|\tau| = g$ that appear in (7.18). For any $h \in [\mathcal{H}]$, there are $\sum_{r=1}^{\min\{K-g, K-\sum_{j=1}^{h-1} K_j\}} \binom{K-r}{g}$ subfile terms $|W_\tau^{(j,h)}|$ in (7.18) for which $|\tau| = g$.

Consequently, since there exist in total $\binom{K}{g}N$ subfiles $W_\tau^{(j,h)}$ with $|\tau| = g$, we have that

$$c_{g,h} = \frac{\sum_{r=1}^{\min\{K-g, K-\sum_{j=1}^{h-1} K_j\}} \binom{K-r}{g}}{\binom{K}{g} N}. \tag{7.26}$$

This implies that our optimal delivery time is lower bounded as

$$T^*(\mathbf{K}) \geq \min_{\mathbf{Z}} \sum_{h=1}^{\mathcal{H}} \sum_{g=0}^{K} \frac{\sum_{r=1}^{\min\{K-g, K-\sum_{j=1}^{h-1} K_j\}} \binom{K-r}{g}}{\binom{K}{g} N} x_{g,l}. \tag{7.27}$$

We recall that the successive refinement source coding applied to the files, implies the following equalities

$$\sum_{g=0}^{K} x_{g,h} = (r_h - r_{h-1})N, \quad \forall h \in [\mathcal{H}] \tag{7.28}$$

while the sum cache size constraint forces

$$\sum_{g=0}^{K} g \cdot \left(\sum_{h=1}^{\mathcal{H}} x_{g,h} \right) \leq KM. \tag{7.29}$$

Finally, by combining (7.27), (7.28), (7.29), the desired lower bound can be derived from the following linear program

$$\underset{x_{g,h}}{\text{minimize}} \ \sum_{h=1}^{\mathcal{H}} \sum_{g=0}^{K} c_{g,h} x_{g,h}$$

$$\begin{aligned} \text{subject to} \;\; & (7.28), (7.29), \\ & x_{g,h} \geq 0, \quad h = 1, ..., \mathcal{H}. \end{aligned}$$

This concludes the proof. □

Chapter 8

Heterogeneous Coded Distributed Computing

In this chapter we address the coded distributed computing problem where each computing node has different fixed computational power, which is directly motivated by practical computing systems that have heterogeneous resources. With the aim of reducing the overall execution time — comprised of mapping, shuffling and reduce time — required to compute a set of arbitrary output functions from a dataset of files, we propose a novel combination of a files assignment strategy, a output functions assignment strategy and a shuffling scheme that exploit the heterogeneity of the nodes' computational powers. In particular, for a fixed heterogeneous assignment of the output functions to the computing nodes, we will prove that, under the reasonable assumption that all the elements of the dataset are replicated the same number of times across the nodes, our files assignment strategy and shuffling scheme jointly achieve a communication load that is optimal within a multiplicative gap of 2. Furthermore, we will also identify a class of heterogeneous distributed computing problems with given computation loads at the nodes (i.e. fixed arbitrary number of files assigned to each node) and given output functions assignment, for which our proposed shuffle scheme is exactly optimal under the aforementioned assumption. Interestingly, some of the results presented in this chapter are strongly related and draw from the techniques that we have developed for the topology-aware shared-cache setting addressed[1] in Chapter 4.

8.1 Introduction

Distributed computing is known to be one of the most important paradigms to speed up large scale data analysis. One of the most well-known frameworks for distributed computing is the so-called *MapReduce* model which decomposes the computation in three distinct phases: the map phase, the shuffle phase and the reduce phase [23]. In

[1] It is important to highlight that we will only present preliminaries contributions on this topic of heterogeneous coded distributed computing and that more complete analysis as well as further extensions of this results are still currently under investigation.

the map phase, computing nodes process part of the data available locally to produce some intermediate values that are then exchanged between the nodes during the shuffle phase, in order to prepare the subsequent reduce phase where the nodes compute the final output results distributedly. Unfortunately, the time that the system spend in the shuffle phase increases rapidly with the number of computing nodes and it does not allow distributed computing to scale with the number of such nodes. To alleviate the impact of the communication load on distributed computing systems, the work in [16] proposed a new variant of this MapReduce framework, called *Coded MapReduce*, which leverages carefully designed redundant computations to enable coding opportunities that substantially reduce the inter-node communication load (i.e. reduce the shuffle time) of distributed computing. In particular, for a MapReduce-type distributed computing system of Λ nodes that have to process a dataset of N files to obtain K output functions, Li et al. showed that the optimal communication load (number of bits communicated by the nodes normalized by the total number of bits required by all the nodes) is given by

$$\tau^*_{CMR}(\gamma, \Lambda) = \frac{1}{\Lambda} \frac{\Lambda - \Lambda\gamma}{\Lambda\gamma}, \tag{8.1}$$

where γ is the size of the fraction of the dataset stored by each computing node. It can be observed that the shuffle scheme that achieves this optimal load is isomorphic to the delivery scheme introduced in [15] for the cache-aided D2D setting. An important limitation of this Coded MapReduce framework is the fact that it considers a homogeneous setting, where all nodes are assigned the same number of files (a fraction γ of the dataset per node) and the same number of output functions ($\frac{K}{\Lambda}$ output functions per node). However, these uniform assignments might not be preferable in an computing system where nodes have heterogeneous resources, since the overall execution time would be dominated by the nodes with less available resources. In fact, it is common to have computing nodes with heterogeneous storage, processing and communication capacities in practical computer clusters (e.g. Amazon EC2 clusters are comprised of heterogeneous computing nodes). Motivated by the above, in the following sections we present new results for the coded distributed computing problem where each node has a different computational power. In particular, we will consider three different scenarios. In the first, we will assume that both the computation load (i.e. number of files of the dataset that each computing node stores) and the reduce load (i.e. number of output functions that each node has to compute) of each node are fixed. On the contrary, in the second scenario we allow the design of the computation loads for given arbitrary reduce loads. Finally, in the third scenario, we assume that both the computation loads and the reduce loads of the nodes are flexible and can be properly designed.

8.1.1 Related Works

Most of the known works on heterogeneous coded distributed computing assume that the computation load of each node is arbitrary and fixed. The first results in this direction can be found in the work in [90], which provides information-theoretically optimal results for the setting with 3 computing nodes having heterogeneous computation

loads, but where the assignment of the output functions is uniform across the nodes. The work in [91] proposes an achievable scheme for any given computation load and any given function assignment at each node. Furthermore, the authors in [91] propose, for any given computation load at each node, two different strategies for the output functions assignment which allow them to show the order-optimality of their achievable communication loads. By means of numerical results, the authors of [91] show that their proposed schemes can have a better communication-computation trade-off than the trade-off of the homogeneous Coded MapReduce in [16]. The problem of file assignment for given computation loads is also addressed in [92], which considers a heterogeneous network of computing nodes consisting of multiple homogeneous networks. For such a problem, the authors prove that their achievable communication load is within a constant multiplicative gap from optimal. A more recent work in [93] considers the fully heterogeneous distributed computing problem with given arbitrary computation loads and reduce loads for which they propose an achievable scheme that requires the solution of a linear program. By numerical simulations, the authors show that their scheme achieves a lower communication load than all the previously mentioned works. Furthermore, they also provide a lower bound on the communication load, which again can be obtained as a solution of a linear program. Other related works can also be found in [94, 95]. Nevertheless, the exact characterization of the optimal computation-communication tradeoff for heterogeneous distributed computing systems remains an open problem. In this chapter, we provide new ideas and results that bring us one more step closer to the solution of such problems.

8.2 Heterogeneous Distributed Computing Model

In this section, we describe a general MapReduce-inspired coded distributed computing problem with heterogeneous computational powers, and define the metric that we will use to assess the goodness of our proposed algorithms.

We consider the problem of computing K arbitrary output functions from a dataset consisting of N files of F bits on a cluster of Λ distributed computing nodes, for some positive integers $K, N, F, \Lambda \in \mathbb{N}$. We assume that each node $\lambda \in [\Lambda]$ has a computational power described by the dimensionless quantity c_λ, such that $\mathbf{c} = (c_1, c_2, \ldots, c_\Lambda)$ denotes the vector of the computational powers of the nodes. Without loss of generality we assume that $c_i \geq c_j$ for $j \geq i$, and we also assume that such computational powers are normalized such that the total computational power of the Λ nodes sums to one, i.e. $\sum_{\lambda=1}^{\Lambda} c_\lambda = 1$. The distributed computation is coordinated by a central controller which, for each $\lambda \in [\Lambda]$, assigns L_λ output functions to computing node λ, such that $\sum_{\lambda=1}^{\Lambda} L_\lambda = K$. We will refer to the vector $\mathbf{L} = (L_1, L_2, \ldots, L_\Lambda)$ as the reduce load vector, which we assume to be sorted in descending order, i.e. $L_i \geq L_j$ for $j \geq i$.

More specifically, given the input files $\mathcal{F} = \{f_1, f_2, \ldots, f_N\} \in \mathbb{F}_{2^F}$, the objective of the computing problem is to evaluate the output values $\mu_1, \mu_2, \ldots, \mu_K$, where $\mu_k = \phi_k(f_1, f_2, \ldots, f_N) \in \mathbb{F}_{2^B}$, $k \in [K]$, for some $B \in \mathbb{N}$, and $\phi_k : (\mathbb{F}_{2^F})^N \longrightarrow \mathbb{F}_{2^B}$ is a function

that maps all the input files to a binary stream of length B. Inspired by the distributed computing framework MapReduce, we decompose the computation as

$$\phi_k(f_1, f_2, \ldots, f_N) = h_k(g_{k,1}(f_1), g_{k,2}(f_2), \ldots, g_{k,N}(f_N)), \ k \in [K] \tag{8.2}$$

where $\mathbf{g}_n \triangleq [g_{1,n}, g_{2,n}, \ldots, g_{K,n}]$ is the function that maps input file f_n into K intermediate values (IVs) $v_{k,n} = g_{k,n}(f_n) \in \mathbb{F}_{2^T}$, $k \in [K]$, for some $T \in \mathbb{N}$, while $h_k : (\mathbb{F}_{2^T})^N \longrightarrow \mathbb{F}_{2^B}$ is the reduce function that maps the IVs $v_{k,1}, v_{k,2}, \ldots, v_{k,N}$ into the output bit stream u_k. The evaluation of the output functions $\{\phi_k\}_{k=1}^K$ as in (8.2) is done in three sequential phases: the map phase, the shuffle phase and the reduce phase. We assume that each phase starts only when the previous one has finished (*sequential* MapReduce).

We assume that a central controller coordinates the computation. Before the computation starts a central controller partition the set of files $\mathcal{F}$ into Λ sets $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_\Lambda$ such that node λ is assigned to the set $\mathcal{M}_\lambda$ such that $M_\lambda \triangleq |\mathcal{M}_\lambda|$. We will refer to $\boldsymbol{\mathcal{M}} = (\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_\Lambda)$ as the file assignment vector. Furthermore, the central controller decides which output values each computing node will have to evaluate. We denote by $\mathcal{L}_\lambda$ the set of the indices of the output values that will be computed by node λ, for some set $\mathcal{L}_\lambda \subseteq [K]$ such that $|\mathcal{L}_\lambda| = L_\lambda$. Hereinafter, the quantity L_λ will be also referred to as the *reduce load* of node λ, $\lambda \in [\Lambda]$. We now proceed with an important definition which tells us the class of file assignments $\boldsymbol{\mathcal{M}}$ considered throughout this chapter.

Definition 4. We say that a file assignment $\boldsymbol{\mathcal{M}}$ is *homogeneous* if all files $f \in \mathcal{F}$ are assigned to the same number of computing nodes.

8.2.1 Map Phase

In this phase, the central controller distributes to node $\lambda, \lambda \in [\Lambda]$ a set of files $\mathcal{M}_\lambda \subseteq \mathcal{F}$ which are mapped by node λ to obtain the intermediate values $\{v_{k,n}\}_{k=1}^K$ for each $f_n \in \mathcal{M}_\lambda$. We define *computation load* of node λ the number of assigned files to node λ, normalized by the total number of files N, i.e. $\gamma_\lambda \triangleq \frac{|\mathcal{M}_\lambda|}{N}$, while the total computation load is $t \triangleq \sum_{\lambda=1}^{\Lambda} \gamma_\lambda$. We will use the symbol $\boldsymbol{\gamma}$ to refer to the vector of the computation loads of the nodes, i.e. $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_\Lambda)$. Furthermore, we assume that node k processes its assigned files with a mapping computational speed of $c_\lambda^{(m)}$ bits per second, where $c_\lambda^{(m)} \triangleq c_\lambda \cdot b_m$, for some $b_m \in \mathbb{R}$. It follows that the time that node λ spends in mapping files in $\mathcal{M}_\lambda$ can be expressed as

$$\mathcal{T}_\lambda^{(m)} = \frac{M_\lambda F}{c_\lambda^{(m)}}, \tag{8.3}$$

thus resulting in a total map time of

$$\mathcal{T}^{(m)} = \max_{\lambda \in [\Lambda]} \left\{ \frac{M_\lambda F}{c_\lambda^{(m)}} \right\} \tag{8.4}$$

bits per second.

8.2.2 Shuffle Phase

After all the nodes have finished mapping their assigned files, the Λ computing nodes exchange some of the computed intermediate values such that at the end of the shuffle phase each node λ will have acquired all intermediate values $v_{k,1}, v_{k,2}, \ldots, v_{k,N}, \forall k \in \mathcal{L}_\lambda$ required for computing output values $\{u_k | k \in \mathcal{L}_\lambda\}$ in the subsequent reduce phase. To do so, each node λ constructs the message $X_\lambda \in \mathbb{F}_{2^{\ell_\lambda}}$ for some $\ell_\lambda \in \mathbb{N}$, as a function of the intermediate values computed in the map phase. In particular, node λ uses some encoding function $\Psi_k : (\mathbb{F}_{2^T})^{K|\mathcal{M}_\lambda|} \longrightarrow \mathbb{F}_{2^{\ell_\lambda}}$ to construct the symbol

$$X_\lambda = \Psi_\lambda(\mathbf{g}_n(f_n) : n \in \mathcal{M}_\lambda),$$

which is later multicasted to all other nodes through a channel of capacity c_s bits per second. We define *communication load* the normalized total number of bits sent through the channel during the shuffle phase as

$$\tau \triangleq \frac{\sum_\lambda^\Lambda \ell_\lambda}{KNT}, \tag{8.5}$$

which corresponds to a shuffle time of

$$\mathcal{T}^{(s)} = \frac{\sum_\lambda^\Lambda \ell_\lambda}{c_s} \tag{8.6}$$

bits per second.

We now proceed with an important definition.

Definition 5. We say that a shuffle scheme is *one-shot* if any computing node can recover each of all its required bits (of the required intermediate values) from the intermediate values computed in the map phase available locally and at most one transmitted message by any other node.

8.2.3 Reduce Phase

In the last phase, node λ uses the received symbols $X_1, X_2, \ldots, X_\Lambda$ and the computed intermediate values $\{\mathbf{g}_n(f_n) : n \in \mathcal{M}_\lambda\}$ to obtain the inputs of the reduce functions $\{h_k : k \in \mathcal{L}_\lambda\}$, i.e. for each $k \in \mathcal{L}_\lambda$ and some encoding function $\chi_\lambda^k = \mathbb{F}_{2^{\ell_1}} \times \mathbb{F}_{2^{\ell_2}} \times \cdots \times \mathbb{F}_{2^{\ell_\Lambda}} \times (\mathbb{F}_{2^T})^{K|\mathcal{M}_\lambda|} \longrightarrow (\mathbb{F}_{2^T})^N$, node λ computes

$$(v_{k,1}, v_{k,2}, \ldots, v_{k,N}) = \chi_\lambda^k(X_1, X_2, \ldots, X_\Lambda, \{\mathbf{g}_n(f_n) : n \in \mathcal{M}_\lambda\}).$$

Finally, node λ computes for each $k \in \mathcal{L}_\lambda$ the reduce function $h_k(v_{k,1}, v_{k,2}, \ldots, v_{k,N})$ to complete its assigned jobs. We assume that the reduce computational speed of node λ is of $c_\lambda^{(r)}$ bits per second, where $c_\lambda^{(r)} = c_\lambda \cdot b_r$, for some $b_r \in \mathbb{R}$. Hence, the time that node λ spends in computing its assigned output values can be expressed as

$$\mathcal{T}_\lambda^{(r)} = \frac{L_\lambda NT}{c_\lambda^{(r)}}, \tag{8.7}$$

thus resulting in the total reduce time

$$\mathcal{T}^{(r)} = \max_{\lambda \in [\Lambda]} \left\{ \frac{L_\lambda NT}{c_\lambda^{(r)}} \right\} \quad \text{bits per second.} \tag{8.8}$$

8.2.4 Problem Formulation

For the heterogeneous coded distributed computing model described above, the metric of interest is the total computing time

$$\mathcal{T}_\Sigma \triangleq \mathcal{T}^{(m)} + \mathcal{T}^{(s)} + \mathcal{T}^{(r)}.$$

The optimal duration of each phase is denoted as $\mathcal{T}^{*(m)}, \mathcal{T}^{*(s)}, \mathcal{T}^{*(r)}$, respectively. In this work, we consider three different scenarios which we describe in the following paragraphs.

Scenario with fixed computation loads and fixed reduce loads

In this scenario we assume that the computation load of each computing node is fixed, i.e. the vector $\boldsymbol{\gamma}$ is fixed and given, which implies that the mapping time $\mathcal{T}^{(m)}$ is fixed and cannot be optimized. At the same time, we assume that also the reduce load vector $\mathbf{L}$ is given, which in turn implies that also the reduce phase time $\mathcal{T}^{(r)}$ is fixed, thus leaving the shuffle time $\mathcal{T}^{(s)}$ the only quantity to be minimize. We notice that minimizing the shuffle time $\mathcal{T}^{(s)}$ is equivalent to minimizing the communication load τ. Hereinafter, we denote $\tau^*(\mathbf{L}, \boldsymbol{\gamma})$ the optimal communication load for any given reduce load vector $\mathbf{L}$ and any given computation load vector $\boldsymbol{\gamma}$. Furthermore, we will use $\tau^*_{hom,os}(\mathbf{L}, \boldsymbol{\gamma})$ to refer to the optimal communication load under the assumptions of homogeneous file assignment (cf. Definition 4) and one-shot shuffling (cf. Definition 5).

Scenario with flexible computation loads and fixed reduce loads

We now consider the scenario where the reduce load vector $\mathbf{L}$ is given and that the vector of the computation loads $\boldsymbol{\gamma}$ can be optimized as a function of $\mathbf{L}$. Nevertheless, we assume that the total computation load to distribute among the nodes equals the value t, i.e. $\sum_{\lambda=1}^{\Lambda} \gamma_\lambda = t$. Unlike the previous scenario, here, we can optimize the time that the system spend in the shuffle and map phase, while the reduce time is fixed. For this scenario, we denote the optimal communication load under the assumption of homogeneous file assignment as $\tau^*(\mathbf{L}, t)$.

Scenario with flexible computation loads and flexible reduce loads

In this third scenario, we assume that both the computation loads and reduce loads can be optimized as a function of the computational power vector $\mathbf{c}$.

8.3 Main Results

In this section we present our main contributions on heterogeneous coded distributed computing for the three considered scenarios described above. Hereinafter, unless explicitly stated, all our results hold for integer values of the total communication load t.

8.3.1 Fixed Computation Loads and Fixed Reduce Loads

We start by presenting a lower bound under the assumption of homogeneous file assignment.

Theorem 11. *For the heterogeneous coded distributed computing problem with given computation load vector $\boldsymbol{\gamma}$ and given reduce load vector $\mathbf{L}$, the optimal communication load, under the assumption of homogeneous file assignment, satisfies*

$$\tau^*_{hom}(\mathbf{L}, \boldsymbol{\gamma}) \geq \frac{\sum_{\lambda \in [\Lambda]} L_\lambda (1 - \gamma_\lambda)}{Kt}, \tag{8.9}$$

where $t = \sum_{\lambda=1}^{\Lambda} \gamma_\lambda$ and $t \in [\Lambda]$.

The proof is relegated to Section 8.6.

Equipped with the above lower bound, we continue by identifying a set of pairs $(\mathbf{L}, \boldsymbol{\gamma})$ for which we can provide an optimal achievable scheme. To this end, we start by defining a computation load vector $\boldsymbol{\gamma}$ given any reduce load vector $\mathbf{L}$.

Definition 6. For any reduce load vector $\mathbf{L}$, any $t \in [\Lambda]$ and any $\lambda \in [\Lambda]$ we define

$$\bar{\gamma_\lambda}(\mathbf{L}) \triangleq \frac{L_\lambda \sum_{\eta \in C_{t-1}^{[\Lambda]\setminus\{\lambda\}}} \prod_{j=1}^{t-1} L_{\eta(j)}}{\sum_{\omega \in C_t^{[\Lambda]}} \prod_{j=1}^{t} L_{\omega(j)}}, \tag{8.10}$$

which in turn defines the vector $\bar{\boldsymbol{\gamma}}(\mathbf{L}) = (\bar{\gamma_1}(\mathbf{L}), \bar{\gamma_2}(\mathbf{L}), \ldots, \bar{\gamma_\Lambda}(\mathbf{L}))$.

Observation 3. *We notice that the expression of the computation load $\bar{\gamma_\lambda}(\mathbf{L})$ defined above coincides with equation* (4.16)*, which tells the normalized memory allocated to cache λ as a function of the cache occupancy vector $\mathbf{L}$ for the topology-aware shared-cache problem discussed in Chapter 4.*

Equipped with definition 6, we are now ready to present our first optimality result of this chapter.

Theorem 12. *For an heterogeneous distributed computing problem with reduce load vector $\mathbf{L}$ and computation load vector $\boldsymbol{\gamma} = \bar{\boldsymbol{\gamma}}(\mathbf{L})$, the optimal communication load $\tau^*_{hom}(\mathbf{L}, \bar{\boldsymbol{\gamma}}(\mathbf{L}))$ under the assumption of homogeneous file assignment is given by*

$$\tau^*_{hom}(\mathbf{L}, \bar{\boldsymbol{\gamma}}(\mathbf{L})) = \frac{\sum_{\lambda \in [\Lambda]} L_\lambda (1 - \bar{\gamma_\lambda}(\mathbf{L}))}{Kt}. \tag{8.11}$$

Proof. For the considered computation load vector $\bar{\boldsymbol{\gamma}}(\mathbf{L})$, which is function of the reduce load vector $\mathbf{L}$, the achievable scheme is presented in Section 8.4, while the matching converse was presented in Theorem 11, which is here used with $\boldsymbol{\gamma} = \bar{\boldsymbol{\gamma}}(\mathbf{L})$. □

Remark 10. *We notice that the achievable scheme presented in Section 8.4 can be seen as a D2D version of the topology-aware scheme for the shared-cache problem presented in Chapter 4, Section 4.4.*

Theorem 12 states the optimality of our achievable scheme, under the aforementioned assumption of homogeneous file assignment, for the pair $(\mathbf{L}, \bar{\boldsymbol{\gamma}}(\mathbf{L}))$ where the computation load $\boldsymbol{\gamma}$ is function of $\mathbf{L}$ (for any $\mathbf{L}$, where $L_\lambda \in \mathbb{N}$) as in Definition 6. The optimal communication load for any pair $(\mathbf{L}, \boldsymbol{\gamma})$ remains an open problem.

We notice that for any non uniform reduce load vector $\mathbf{L}$, the achievable communication load for the tuple $(\mathbf{L}, \bar{\boldsymbol{\gamma}}(\mathbf{L}))$ is lower than the communication load $\tau^*_{CMR}(\Lambda, \gamma)$ in equation (8.1) by the standard coded MapReduce algorithm which uses $\mathbf{L} = (\frac{K}{\Lambda}, \ldots, \frac{K}{\Lambda})$ and $\boldsymbol{\gamma} = (\frac{t}{\Lambda}, \ldots, \frac{t}{\Lambda})$. While this implies that if we want to reduce the communication load we have to choose a reduce load vector $\mathbf{L}$ as skewed as possible, we also notice that, in the scenario where all nodes have the same computational power, a skewed (non uniform) $\mathbf{L}$ implies both a higher map time and reduce time than the case when the reduce load vector is selected to be uniform, i.e. $\mathbf{L} = (\frac{K}{\Lambda}, \ldots, \frac{K}{\Lambda})$. Therefore, it is evident that in order to minimize the overall computing time (comprised of map time, shuffle time and reduce time) we have to take into account the specific computing problem which will affect the time that each node will spend in mapping a file of the dataset as well as the time required to produce a given output value in the reduce phase. When we will consider the scenario where both the computation load vector and the reduce load vector can be optimized we will present a strategy that aims at minimizing the reduce time while also reducing the shuffle time.

8.3.2 Flexible Computation Loads and Fixed Reduce Loads

In the next theorem we present an order optimality result for the considered scenario where the computation load vector $\boldsymbol{\gamma}$ can be optimized as a function of the given reduce load vector $\mathbf{L}$ in order to minimize the communication load τ.

Theorem 13. *For an heterogeneous distributed computing problem with reduce load vector $\mathbf{L}$, the optimal communication load $\tau^*_{hom,os}(\mathbf{L}, t)$ under the assumptions of homogeneous file assignment and one-shot shuffling satisfies*

$$\frac{1}{2} \cdot \tau^*_{hom}(\mathbf{L}, \bar{\boldsymbol{\gamma}}(\mathbf{L})) \leq \tau^*_{hom,os}(\mathbf{L}, t) \leq \tau^*_{hom}(\mathbf{L}, \bar{\boldsymbol{\gamma}}(\mathbf{L})), \tag{8.12}$$

where $\tau^*_{hom}(\mathbf{L}, \bar{\boldsymbol{\gamma}}(\mathbf{L}))$ is provided in Theorem 12, for some $t \in [\Lambda]$.

Proof. The converse bound and derivation of the optimality gap for Theorem 13 are presented in Section 8.5, while the achievable scheme is the one that achieves the communication load in Theorem 12 and it is presented in Section 8.4. □

Theorem 13 tells us that an order-optimal selection of the computation load vector $\boldsymbol{\gamma}$, given a total computation load t, is the one described by the vector $\bar{\boldsymbol{\gamma}}(\mathbf{L})$. This strategy allocates more computation load to those computing nodes that have to evaluate more output functions, in a way that in the shuffle phase a coding gain of t is always achievable regardless of $\mathbf{L}$. At the same time, allocating more computation load to those nodes that are more loaded in the reduce phase allows to simultaneously reduce also the total number of intermediate values that have to be exchanged among the nodes, i.e. such approach allows to get an higher "local caching gain"[2] than the one that would be achieved with a uniform distribution of the total computation load among the nodes.

Equipped with an order-optimal solution for reducing the communication load for a setting with a non uniform reduce load vector $\mathbf{L}$, in the next paragraph we show a simple yet effective way to select such reduce load vector $\mathbf{L}$ as a function of the computational power vector $\mathbf{c}$.

8.3.3 Flexible Computation Loads and Flexible Reduce Loads

In this scenario the computation load vector $\boldsymbol{\gamma}$ and the reduce load vector $\mathbf{L}$ can be selected as a function of the computational power vector $\mathbf{c}$ in order to minimize the overall computing time $\mathcal{T}_{\Sigma}$. To this end, we take a separation approach and we first focus on minimizing the reduce time $\mathcal{T}^{(r)}$, before proceeding with the minimization of the shuffle time $\mathcal{T}^{(s)}$.

The optimal reduce time, minimized over all possible reduce load vectors $\mathbf{L} \in \mathbb{N}^{\Lambda}$, can be evaluated as

$$\mathcal{T}^{*(r)} = \min_{\mathbf{L}\in\mathbb{N}^{\Lambda}} \max_{\lambda\in[\Lambda]} \left\{ \frac{L_\lambda NT}{c_\lambda^{(r)}} \right\} = \frac{NT}{b_r} \min_{\mathbf{L}\in\mathbb{N}^{\Lambda}} \max_{\lambda\in[\Lambda]} \left\{ \frac{L_\lambda}{c_\lambda} \right\}, \tag{8.13}$$

which it can be seen that is minimized for $L_\lambda = Kc_\lambda$. Therefore, the minimum reduce time takes the value

$$\mathcal{T}^{*(r)} = \frac{KNT}{b_r}. \tag{8.14}$$

With the optimal reduce load vector $\mathbf{L}^*(\mathbf{c}) \triangleq K\mathbf{c}$ at hand, we can proceed to minimizing the shuffle time $\mathcal{T}^{(s)}$ as a function of $\mathbf{L}^*(\mathbf{c})$, which in turn is equivalent to minimizing the communication load $\tau(K\mathbf{c}, t)$. The following corollary holds.

Corollary 2. *For the coded distributed computing problem with computational power vector* $\mathbf{c}$ *and optimal reduce load vector* $\mathbf{L}^*(\mathbf{c}) = K\mathbf{c}$*, the optimal communication load, under the assumptions of homogeneous file assignment and one-shot shuffling, satisfies*

$$\frac{1}{2} \cdot \tau^*_{hom}(K\mathbf{c}, \bar{\boldsymbol{\gamma}}(K\mathbf{c})) \leq \tau^*_{hom,os}(K\mathbf{c}, t) \leq \tau^*_{hom}(K\mathbf{c}, \bar{\boldsymbol{\gamma}}(K\mathbf{c})) \tag{8.15}$$

Proof. Corollary 2 follows directly from theorem 13. □

[2]Here, the term local caching gain is borrowed from the coded caching literature.

Therefore, in this scenario where we can select the output functions assignment (i.e. the reduce loads of the nodes) as well as the computation loads of each node as a function of the computational powers, selecting the reduce load vector $\mathbf{L}^*(\mathbf{c}) = K\mathbf{c}$ minimizes the reduce time and allows for an order-optimal shuffle time, achieved by the algorithm described in Section 8.4. This solution has the drawback to slightly increase the map time $\mathcal{T}^{(m)}$ compared to the case where all computing nodes have the same storage size. In particular, for the optimal choice of the reduce load vector $\mathbf{L}^*(\mathbf{c}) = K\mathbf{c}$, the map time takes the value

$$\mathcal{T}^{(m)} = \max_{\lambda \in [\Lambda]} \left\{ \frac{\bar{\gamma_\lambda}(K\mathbf{c})NF}{c_\lambda^{(m)}} \right\}, \tag{8.16}$$

which is in general higher than the map time given by a uniform computation load vector $\boldsymbol{\gamma} = (\frac{t}{\Lambda}, \ldots, \frac{t}{\Lambda})$. However, we recall that in practical scenarios the time spent by the system in the map phase is much smaller than the shuffle time and the reduce time (cf. [16]), therefore this increased map time does not cancel-out the benefits of our proposed solution which aims at minimizing the more dominant reduce time and shuffle time.

8.4 A Novel File Assignment and Shuffle Scheme for Heterogeneous Coded Distributed Computing

In this section we present a file assignment and shuffle algorithm for the heterogeneous distributed computing problem with reduce load vector $\mathbf{L}$ and computation load vector $\bar{\boldsymbol{\gamma}}(\mathbf{L})$ (cf. Definition 6), for any total computation load $t \in [\Lambda]$. The reader will notice that the described file assignment and shuffle schemes can be seen as the D2D versions of the cache placement and delivery scheme described in Section 4.4, applied to the considered distributed computing problem.

8.4.1 File Assignment Scheme

For any given $t \in [\Lambda]$ and any given reduce load vector $\mathbf{L}$, which in turn imply the computation load vector $\bar{\boldsymbol{\gamma}}(\mathbf{L})$, the central controller splits the dataset $\mathcal{F}$ into N files $f_1, f_2, \ldots, f_N$ of F bits, where N takes the value[3]

$$N = \sum_{\eta \in C_t^{[\Lambda]}} \prod_{j=1}^{t} L_{\eta(j)}, \tag{8.17}$$

and where we recall that $C_t^{[\Lambda]}$ is the set of all possible t-combinations of the set $[\Lambda]$. Let us consider a bijection that maps the set of indices $[N] = \{1, 2, \ldots, N\}$ to the set

$$\Delta_{\mathbf{L},t} \triangleq \left\{ (\eta, 1), (\eta, 2), \ldots, (\eta, |\mathcal{A}_\eta|) | \eta \in C_t^{[\Lambda]} \right\},$$

[3]The scheme can be easily extended to work also for N taking any multiplicative value of the RHS of equation (8.17).

where $\mathcal{A}_\eta \triangleq \left\{1, 2, \ldots, \prod_{j=1}^{t} L_{\eta(j)}\right\}$. Hereinafter, we will use the above two sets $[N]$ and $\Delta_{\mathbf{L},t}$ interchangeably, in order to facilitate the exposition of the algorithm. Thus, for example, we can now rewrite the set of files $\{f_n\}_{n=1}^{N}$ as

$$\left\{f_{(\eta,1)}, f_{(\eta,2)}, \ldots, f_{(\eta,|\mathcal{A}_\eta|)} \mid \eta \in C_t^{[\Lambda]}\right\}, \quad \forall \lambda \in [\Lambda].$$

Afterwards, the central controller assigns to each node $\lambda \in [\Lambda]$ all files $f_{(\eta,m_\eta)}, m_\eta \in \mathcal{A}_\eta$ whose first superscript η includes λ, which in turn results in the following file assignment

$$\mathcal{M}_\lambda = \left\{f_{(\eta,m_\eta)} | \eta \ni \lambda, m_\eta \in \mathcal{A}_\eta\right\}.$$

It can be verified that the described file assignment is congruent with the computation load vector $\bar{\gamma}(\mathbf{L})$. Once the file assignment is done, each node proceed to map each assigned files to produce all the corresponding intermediate values.

8.4.2 Shuffle Scheme

After the map phase, each node λ will have obtained for each file $f_{(\eta,m_\eta)}, \eta \ni \lambda, m_\eta \in \mathcal{A}_\eta$ the K intermediate values $\{v_{1,(\eta,m_\eta)}, v_{2,(\eta,m_\eta)}, \ldots, v_{K,(\eta,m_\eta)}\}$. We recall that at the end of the shuffle phase each node λ needs to have acquired the set of intermediate values

$$\{v_{k,n} | k \in \mathcal{L}_\lambda, n \in [N]\}.$$

We proceed with the following instrumental proposition.

Proposition 5. *For any $t+1$-tuple $\mathcal{Q} \subset [\Lambda]$, the total number of intermediate values $\{v_{k,(\eta,m_\eta)}\}$ with $\eta = \mathcal{Q}\backslash\{\lambda\}$ that are missing from computing node $\lambda, \lambda \in \mathcal{Q}$, is independent of λ and it equals*

$$P_{\mathcal{Q}} \triangleq \prod_{j=1}^{t+1} L_{\mathcal{Q}(j)}. \tag{8.18}$$

Proof. For any $t+1$-tuple $\mathcal{Q} \subset [\Lambda]$, let us consider node $\lambda \in \mathcal{Q}$. For any $k \in \mathcal{L}_\lambda$, computing node λ has to receive from the other nodes $\prod_{j=1}^{t} L_{\eta(j)}$ intermediate values $\{v_{k,(\eta,m_\eta)}\}$ with $\eta = \mathcal{Q} \setminus \{\lambda\}$ since $m_\eta \in \mathcal{A}_\eta$ and $|\mathcal{A}_\eta| = \prod_{j=1}^{t} L_{\eta(j)}$. This in turn means that, since $|\mathcal{L}_\lambda| = L_\lambda$, the total number of intermediate values that need to be sent to node λ is $L_\lambda \prod_{j=1}^{t} L_{\eta(j)} = \prod_{j=1}^{t+1} L_{\mathcal{Q}(j)}$, which does not depend on λ. □

For any $t+1$-tuple $\mathcal{Q} \subseteq [\Lambda]$ and any $\lambda \in \mathcal{Q}$, we set $\eta = \mathcal{Q} \setminus \{\lambda\}$ and relabel the set of intermediate values $\left\{v_{k,(\eta,m_\eta)} | k \in \mathcal{L}_\lambda, m_\eta \in \mathcal{A}_\eta\right\}$ as $\mathcal{W}_\mathcal{Q} = \left\{w_{(\eta,j)}^{(\lambda)} | j \in [P_\mathcal{Q}]\right\}$. Then, we split each intermediate value $w_{(\eta,j)}^{(\lambda)}$ into t parts as $w_{(\eta,j)}^{(\lambda)} = \{w_{(\eta,j,p)}^{(\lambda)} | p \in \eta\}$ such that $|w_{(\eta,j,p)}^{(\lambda)}| = \frac{T}{t}$. At this point, for any $t+1$-tuple $\mathcal{Q}$ and any $j \in [P_\mathcal{Q}]$ we construct the following $t+1$ messages

$$X_{\lambda,\mathcal{Q}\backslash\{\lambda\},j} = \bigoplus_{\lambda' \in \mathcal{Q}\backslash\{\lambda\}} w_{(\mathcal{Q}\backslash\{\lambda'\},j,\lambda)}^{(\lambda')} \quad \forall \lambda \in \mathcal{Q}, \tag{8.19}$$

where $X_{\lambda,\mathcal{Q}\setminus\{\lambda\},j}$ is a message transmitted by node λ to all nodes in the set $\mathcal{Q}\setminus\{\lambda\}$. It can be verified that, for any $\lambda \in \mathcal{Q}$, each intended node $\lambda' \in \mathcal{Q}\setminus\{\lambda\}$ can recover its required intermediate value part $w^{(\lambda')}_{(\mathcal{Q}\setminus\{\lambda'\},j,\lambda)}$ from $X_{\lambda,\mathcal{Q}\setminus\{\lambda\},j}$ and the intermediate values computed in the map phase. This decoding proceedure is analogous to the one of the standard Coded MapReduce in [16].

8.4.3 Communication Load

For any $\mathcal{Q} \in C^{[\Lambda]}_{t+1}$ and any $j \in [P_{\mathcal{Q}}]$, the number of transmitted messages is $t+1$, thus resulting in a total number of transmissions of

$$(t+1)\cdot \sum_{\mathcal{Q}\in C^{[\Lambda]}_{t+1}} P_{\mathcal{Q}}. \tag{8.20}$$

Then, we notice that the number of bits of each transmitted message is equal to $|X_{\lambda,\mathcal{Q}\setminus\{\lambda\},j}| = \frac{T}{t}$, which is the same for all transmissions. It follows that the communication load achieved by our proposed scheme can be written as

$$\begin{aligned}
\tau(\mathbf{L},\bar{\boldsymbol{\gamma}}(\mathbf{L})) &= \frac{(t+1)\sum_{\mathcal{Q}\in C^{[\Lambda]}_{t+1}} P_{\mathcal{Q}}\frac{T}{t}}{KNT} && (8.21)\\
&= \frac{(t+1)\sum_{\eta\in C^{[\Lambda]}_{t+1}}\prod_{j=1}^{t+1} L_{\eta(j)}}{tK\sum_{\omega\in C^{[\Lambda]}_{t}}\prod_{j=1}^{t} L_{\omega(j)}} && (8.22)\\
&= \frac{(t+1)e_{t+1}(\mathbf{L})}{tKe_t(\mathbf{L})} && (8.23)\\
&= \frac{\sum_{\lambda\in[\Lambda]} L_\lambda(1-\bar{\gamma_\lambda}(\mathbf{L}))}{Kt}. && (8.24)
\end{aligned}$$

where in (8.22) we have used (8.17) and where (8.24) follows from simple mathematical manipulations analogous to those in equation (4.19).

8.5 Converse Bound for the Scenario with Given Reduce Loads and Proof of Optimality Gap

In the first part of this section we construct the bound on the communication load $\tau^*_{hom,os}(\mathbf{L},t)$ that along with the achievable scheme presented in Section 8.4 will allow to prove Theorem 13 in the second part of this section (Section 8.5.2).

8.5.1 Lower Bound

We start by recalling that the communication load $\tau(\mathbf{L},t)$ is defined as

$$\tau(\mathbf{L},t) = \frac{\sum_\lambda^\Lambda \ell_\lambda}{KNT} = \frac{\sum_\lambda^\Lambda H(X_\lambda)}{KNT}, \tag{8.25}$$

where $H(\cdot)$ is the entropy function and X_λ is the message transmitted by node λ in the shuffle phase.

Observation 4. *We observe that, under the assumption of one-shot shuffling, the shuffle phase can be seen to be composed of Λ shared-link channels, where in each such channel we have that node λ acts as a transmitter that serves the remaining $\Lambda-1$ computing nodes $[\Lambda]\setminus\{\lambda\}$. In particular, the transmitting node λ uses the intermediate values obtained in the map phase to transmit to all other nodes their desired intermediate values.*

The above observation is crucial for the derivation of the converse bound, which relies on the aforementioned one-shot assumption. This same idea of seeing, under the one-shot assumption, the D2D shard-link setting as a set of shared-link BC settings was already used in [96] for the D2D coded caching problem.

First, we need to introduce some useful notation. Following the same notation as in [16], we denote by $V_{k,n}$ an i.i.d. random variable uniformly distributed on $\mathbb{F}_{2^T}$ for any $k\in[K]$ and $n\in[N]$ and we let the intermediate values $v_{k,n}$ be the realization of $V_{k,n}$. For some $k\in[K]$ and $\mathcal{N}\subseteq[N]$, we define

$$V_{k,\mathcal{N}} \triangleq \left\{V_{k,n} : n\in(\cap_{\lambda\in\mathcal{N}}\mathcal{M}_\lambda)\setminus\left(\cup_{\lambda'\notin\mathcal{N}}\mathcal{M}_{\lambda'}\right)\right\}.$$

We recall that we restrict our attention to those file assignments that are homogeneous, as defined in Definition 4. This implies that the cardinality of $\mathcal{N}$ must be $|\mathcal{N}|=t$, for $t\in[\Lambda]$. Furthermore, we will use $V^{(\lambda)}_{k,\mathcal{N}}$ to refer to the random variables in $V_{k,\mathcal{N}}$, whose realizations correspond to the intermediate values transmitted by node λ during the shuffle phase. We also denote the number of files that are exclusively mapped by the nodes in set S as

$$a_{S,\boldsymbol{\mathcal{M}}} \triangleq \sum_{\mathcal{J}\subseteq S} |(\underset{q\in\mathcal{J}}{\cap}\mathcal{M}_q)\setminus(\underset{i\notin\mathcal{J}}{\cup}\mathcal{M}_i)|.$$

Finally, for any set χ, we denote the set of all the permutations of the set χ as S_χ.

We now proceed with the proof. Because of the one-shot assumption, we can first focus, for any $\lambda\in[\Lambda]$, on the channel that sees computing node λ to be a transmitter serving the remaining nodes $[\Lambda]\setminus\{\lambda\}$. For such channel we will provide a lower bound on $H(X_\lambda)$. After that, we will sum over all possible $\lambda\in[\Lambda]$ to obtain a lower bound on the total number of bits that must be exchanged in the shuffle phase. We start with an important instrumental lemma which provides a lower bound on $H(X_\lambda)$.

Lemma 9.

$$H(X_\lambda)\geq \sum_{i\in[\Lambda]\setminus\{\lambda\}}\sum_{q\in\mathcal{L}_{\sigma(i)}}\sum_{\eta_i\in C_{t-1}^{[\Lambda]\setminus\{\lambda,\sigma(1),\ldots,\sigma(i)\}}} H(V^{(\lambda)}_{q,\{\lambda,\eta_i\}}), \quad \textit{for some } \sigma\in S_{[\Lambda]\setminus\{\lambda\}}. \tag{8.26}$$

Proof. The above lemma is an adaptation of Lemma 5 of Chapter 3 to the distributed computing shared-link problem where computing node λ serves the remaining $[\Lambda]\setminus\{\lambda\}$

nodes. The lemma can be obtained by first modelling the considered problem as an index coding problem and then applying the single-antenna version of the MAIS bound reported in Lemma 1, similarly as it was done in [52]. Notice that, unlike Lemma 5, the third summation in (8.26) is over only sets η_i of cardinality $t-1$ because of the homogeneous file assignment assumption that forces the second lower index of $V^{(\lambda)}_{k,\mathcal{N}}$ to be of size t. □

We immediately notice that, for any $\sigma \in S_{[\Lambda]\setminus\{\lambda\}}$, (8.26) can be simplified to

$$H(X_\lambda) \geq \underbrace{\sum_{i\in[\Lambda]\setminus\{\lambda\}} \sum_{\eta_i \in C_{t-1}^{[\Lambda]\setminus\{\lambda,\sigma(1),\ldots,\sigma(i)\}}} L_{\sigma(i)} \cdot H(V^{(\lambda)}_{k^*,\{\lambda,\eta_i\}})}_{\tau^{lb}_{\lambda,\sigma}}, \tag{8.27}$$

where k^* refers to *any* $k \in [K]$.

Following the same steps as the converse bound in Section 4.5, we lower bound $H(X_\lambda)$ as

$$H(X_\lambda) \geq \frac{1}{\sum_{\sigma\in S_{[\Lambda]\setminus\{\lambda\}}} Q_\sigma} \sum_{\sigma\in S_{[\Lambda]\setminus\{\lambda\}}} Q_\sigma \tau^{lb}_{\lambda,\sigma}, \tag{8.28}$$

where Q_σ is chosen to be $Q_\sigma = \prod_{j=1}^{t-1} L_{\sigma(\Lambda-t+j)}$. After denoting the RHS of (8.28) as τ^{lb}_λ and recalling that $e_k(\chi)$ denotes the k-th elementary symmetric function in the set χ (see Definition 3), the following lemma holds.

Lemma 10.

$$\tau^{lb}_\lambda = \frac{e_t(\mathbf{L}\setminus\{L_\lambda\})}{e_{t-1}(\mathbf{L}\setminus\{L_\lambda\})} \sum_{\eta\in C_{t-1}^{[\Lambda]\setminus\{\lambda\}}} H(V^{(\lambda)}_{k^*,\{\lambda,\eta\}}) \tag{8.29}$$

Proof. The proof follows directly from Section 4.5 after imposing homogeneous cache placement (see also [51]). □

Next, summing the loads of each shared-link channel it yields

$$\begin{aligned}
\sum_{\lambda=1}^{\Lambda} H(X_\lambda) &\geq \sum_{\lambda=1}^{\Lambda} \frac{e_t(\mathbf{L}\setminus\{L_\lambda\})}{e_{t-1}(\mathbf{L}\setminus\{L_\lambda\})} \sum_{\eta\in C_{t-1}^{[\Lambda]\setminus\{\lambda\}}} H(V^{(\lambda)}_{k^*,\{\lambda,\eta\}}) \\
&= \sum_{\eta\in C_t^{[\Lambda]}} \sum_{\lambda\in\eta} \frac{e_t(\mathbf{L}\setminus\{L_\lambda\})}{e_{t-1}(\mathbf{L}\setminus\{L_\lambda\})} H(V^{(\lambda)}_{k^*,\eta}) \\
&\overset{(a)}{\geq} \sum_{\eta\in C_t^{[\Lambda]}} \frac{e_t(\mathbf{L}\setminus\{L_{\lambda^*(\eta)}\})}{e_{t-1}(\mathbf{L}\setminus\{L_{\lambda^*(\eta)}\})} \sum_{\lambda\in\eta} H(V^{(\lambda)}_{k^*,\eta}) \\
&\overset{(b)}{\geq} \sum_{\eta\in C_t^{[\Lambda]}} \frac{e_t(\mathbf{L}\setminus\{L_{\lambda^*(\eta)}\})}{e_{t-1}(\mathbf{L}\setminus\{L_{\lambda^*(\eta)}\})} H(V_{k^*,\eta})
\end{aligned} \tag{8.30}$$

$$\overset{(c)}{\geq} \sum_{\eta \in C_t^{[\Lambda]}} \frac{e_t(\mathbf{L} \setminus \{L_{\lambda^*(\eta^*)}\})}{e_{t-1}(\mathbf{L} \setminus \{L_{\lambda^*(\eta^*)}\})} T a_{\eta,\mathcal{M}}$$
$$= T \frac{e_t(\mathbf{L} \setminus \{L_{\lambda^*(\eta^*)}\})}{e_{t-1}(\mathbf{L} \setminus \{L_{\lambda^*(\eta^*)}\})} \sum_{\eta \in C_t^{[\Lambda]}} a_{\eta,\mathcal{M}}, \tag{8.31}$$

where in (a) we have used $\lambda^*(\eta) = \text{argmin}_{\lambda \in \eta} \frac{e_t(\mathbf{L} \setminus \{L_\lambda\})}{e_{t-1}(\mathbf{L} \setminus \{L_\lambda\})}$, (b) follows from the fact that $\sum_{\lambda \in \eta} H(V_{k^*,\eta}^{(\lambda)}) \geq H(V_{k^*,\eta})$, and where in (c) we have used $\eta^* = \text{argmin}_{\eta \in C_t^{[\Lambda]}} \frac{e_t(\mathbf{L} \setminus \{L_{\lambda^*(\eta)}\})}{e_{t-1}(\mathbf{L} \setminus \{L_{\lambda^*(\eta)}\})}$ as well as $H(V_{k^*,\eta}) = T a_{\eta,\mathcal{M}}$.

We now observe that the homogeneous file assignment assumption yields

$$\sum_{\eta \in C_t^{[\Lambda]}} a_{\eta,\mathcal{M}} = N. \tag{8.32}$$

Combining (8.31) and (8.32) we have

$$\sum_{\lambda=1}^{\Lambda} H(X_\lambda) \geq TN \frac{e_t(\mathbf{L} \setminus \{L_{\lambda^*(\eta^*)}\})}{e_{t-1}(\mathbf{L} \setminus \{L_{\lambda^*(\eta^*)}\})}. \tag{8.33}$$

We now proceed with the following instrumental lemma.

Lemma 11. *For any $\lambda_1 \in [\Lambda]$ and any $\lambda_2 \in [\Lambda]$, such that $L_{\lambda_1} \geq L_{\lambda_2}$ the following inequality holds*

$$\frac{e_t(\mathbf{L} \setminus \{L_{\lambda_1}\})}{e_{t-1}(\mathbf{L} \setminus \{L_{\lambda_1}\})} \leq \frac{e_t(\mathbf{L} \setminus \{L_{\lambda_2}\})}{e_{t-1}(\mathbf{L} \setminus \{L_{\lambda_2}\})}. \tag{8.34}$$

Proof. Proving (8.34) is equivalent to prove

$$e_t(\mathbf{L} \setminus \{L_{\lambda_1}\}) e_{t-1}(\mathbf{L} \setminus \{L_{\lambda_2}\}) \leq e_t(\mathbf{L} \setminus \{L_{\lambda_2}\}) e_{t-1}(\mathbf{L} \setminus \{L_{\lambda_1}\}) \tag{8.35}$$

Applying the Property 1 in Appendix B.1 to each element of the above expression and moving the RHS of (8.35) to the LHS we can write the above inequality as

$$\begin{aligned}
&[L_{\lambda_2} e_{t-1}(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\}) + e_t(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\})] \cdot \\
&\cdot [L_{\lambda_1} e_{t-2}(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\}) + e_{t-1}(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\})] + \\
&- [L_{\lambda_1} e_{t-1}(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\}) + e_t(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\})] \cdot \\
&\cdot [L_{\lambda_2} e_{t-2}(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\}) + e_{t-1}(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\})] \leq 0
\end{aligned} \tag{8.36}$$

We can see that (8.36) is equivalent to

$$(L_{\lambda_2} - L_{\lambda_1}) \left(e_{t-1}^2(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\}) - e_t(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\}) e_{t-2}(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\}) \right) \leq 0. \tag{8.37}$$

which we have to prove to be correct. By assumption we have that $L_{\lambda_2} - L_{\lambda_1} \leq 0$, while from the Netwon-Maclaurin bound [63] we

$$e_{t-1}^2(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\}) - e_t(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\}) e_{t-2}(\mathbf{L} \setminus \{L_{\lambda_1}, L_{\lambda_2}\}) \geq 0$$

This shows that (8.37) holds, thus completing the proof.

□

Recalling that $\mathbf{L}$ is sorted in decreasing order, Lemma 11 directly implies that $\lambda^*(\eta^*) = 1$. By using this in equation (8.33) we get

$$\tau^*(\mathbf{L}, t, \boldsymbol{\mathcal{M}}) = \frac{\sum_{\lambda=1}^{\Lambda} H(X_\lambda)}{KNT} \tag{8.38}$$

$$\geq \frac{e_t(\mathbf{L} \setminus \{L_1\})}{K e_{t-1}(\mathbf{L} \setminus \{L_1\})}, \tag{8.39}$$

where we have used $\tau^*(\mathbf{L}, t, \boldsymbol{\mathcal{M}})$ to denote the optimal communication load for a given reduce load vector $\mathbf{L}$ and file assignment $\boldsymbol{\mathcal{M}}$ that corresponds to a total computation load t. Finally, the optimal communication load $\tau^*_{hom,os}(\mathbf{L}, t)$, under the aforementioned assumptions of homogeneous file assignment and one-shot shuffling, satisfies

$$\tau^*_{hom,os}(\mathbf{L}, t) \triangleq \min_{\substack{\boldsymbol{\mathcal{M}}: \\ |\mathcal{M}_1|+\cdots+|\mathcal{M}_\Lambda|=tN}} \tau^*(\mathbf{L}, t, \boldsymbol{\mathcal{M}}) \tag{8.40}$$

$$\geq \frac{e_t(\mathbf{L} \setminus \{L_1\})}{K e_{t-1}(\mathbf{L} \setminus \{L_1\})}. \tag{8.41}$$

8.5.2 Optimality Gap

We now proceed to prove that the achievable communication load in (8.24) of our proposed scheme is optimal within a gap of two. Recalling that the achievable communication load is

$$\tau(\mathbf{L}, t) = \frac{t+1}{t} \frac{e_{t+1}(\mathbf{L})}{K e_t(\mathbf{L})}, \tag{8.42}$$

we have

$$\frac{\tau(\mathbf{L}, t)}{\tau^*_{hom,os}(\mathbf{L}, t)} \leq \frac{t+1}{t} \frac{e_{t+1}(\mathbf{L})}{e_t(\mathbf{L})} \frac{e_{t-1}(\mathbf{L} \setminus \{L_1\})}{e_t(\mathbf{L} \setminus \{L_1\})} \tag{8.43}$$

$$\overset{(a)}{=} \frac{t+1}{t} \frac{e_{t+1}(\mathbf{L} \setminus \{L_1\})}{e_t(\mathbf{L})} \frac{e_{t-1}(\mathbf{L} \setminus \{L_1\})}{e_t(\mathbf{L} \setminus \{L_1\})} + \frac{t+1}{t} \frac{L_1 e_t(\mathbf{L} \setminus \{L_1\})}{e_t(\mathbf{L})} \frac{e_{t-1}(\mathbf{L} \setminus \{L_1\})}{e_t(\mathbf{L} \setminus \{L_1\})} \tag{8.44}$$

$$\overset{(b)}{\leq} \frac{t+1}{t} \left(\frac{e_t^2(\mathbf{L} \setminus \{L_1\})}{e_t(\mathbf{L}) e_t(\mathbf{L} \setminus \{L_1\})} + L_1 \frac{e_{t-1}(\mathbf{L} \setminus \{L_1\})}{e_t(\mathbf{L})} \right) \tag{8.45}$$

$$= \frac{t+1}{t} \left(\frac{e_t(\mathbf{L} \setminus \{L_1\}) + L_1 e_{t-1}(\mathbf{L} \setminus \{L_1\})}{e_t(\mathbf{L})} \right) \tag{8.46}$$

$$\overset{(c)}{=} \frac{t+1}{t} \tag{8.47}$$

$$\leq 2, \tag{8.48}$$

where (a) and (c) follow from Property 1 in Appendix B.1 and where (b) follows from the Newton-MacLaurin bound of the elementary symmetric functions [63].

8.6 Converse Bound for the Scenario with Given Computation and Reduce Loads

In this section we provide the proof of Theorem 11. We start with some useful notation. For a subset $S \subseteq [\Lambda]$ and a file assignment $\boldsymbol{\mathcal{M}}$, we denote the number of files that are exclusively mapped by j nodes in S as

$$a^{(j)}_{S,\boldsymbol{\mathcal{M}}} \triangleq \sum_{\mathcal{J}\subseteq S:|S|=j} |(\underset{q\in\mathcal{J}}{\cap}\mathcal{M}_q)\backslash(\underset{i\notin\mathcal{J}}{\cup}\mathcal{M}_i)|,$$

which in turn can be rewritten as

$$a^{(j)}_{S,\boldsymbol{\mathcal{M}}} = \frac{1}{j}\sum_{\lambda\in S} a^{(j,\lambda)}_{S,\boldsymbol{\mathcal{M}}} \tag{8.49}$$

where $a^{(j,\lambda)}_{S,\boldsymbol{\mathcal{M}}}$ is the number of files that are exclusively mapped by j nodes in S including node λ. It holds that

$$\sum_{j=1}^{|S|} a^{(j,\lambda)}_{[\Lambda],\boldsymbol{\mathcal{M}}} = \gamma_\lambda N,\ \lambda \in [\Lambda]. \tag{8.50}$$

Furthermore, for some $\mathcal{Q} \subseteq [K]$ and $\mathcal{N} \subseteq [N]$, we define

$$V_{\mathcal{Q},\mathcal{N}} \triangleq \{V_{q,n} : q \in \mathcal{Q}, n \in \mathcal{N}\},$$

where $V_{q,n}$ was introduced at the beginning of the previous section and it represents the random variable whose realization gives the intermediate value $v_{q,n}$.

Next, for a subset $S \subseteq [\Lambda]$, we define

$$X_S \triangleq (X_\lambda : \lambda \in S),$$

as well as

$$Y_S \triangleq (V_{\mathcal{L}_S,:}, V_{:,\mathcal{M}_S}),$$

where $\mathcal{L}_S = \cup_{\lambda\in S}\mathcal{L}_\lambda$, $\mathcal{M}_S = \cup_{\lambda\in S}\mathcal{M}_\lambda$, and where we use ":" to denote the set of all possible indices. Y_S contains all the intermediate values required by the nodes in S and all those known locally by the nodes in S after the map phase.

We can now start the proof by presenting an important lemma which is instrumental for the derivation of the lower bound.

Lemma 12. *For any file assignment $\boldsymbol{\mathcal{M}}$ and any reduce load vector $\mathbf{L}$, it holds that*

$$H(X_S|Y_{S^c}) \geq T\sum_{\lambda\in S}\sum_{j=1}^{|S|} a^{(j,\lambda)}_{S,\boldsymbol{\mathcal{M}}} \frac{\sum_{q\in S} L_q - j\cdot L_\lambda}{j^2} \quad S \subseteq [\Lambda] \tag{8.51}$$

Proof. The proof of Lemma 12 is presented in Appendix D.1. □

We now proceed to lower bound the optimal communication load $\tau^*(\mathbf{L}, \boldsymbol{\gamma})$ as follows:

$$\tau^*(\mathbf{L}, \boldsymbol{\gamma}) \triangleq \min_{\boldsymbol{\mathcal{M}}} \frac{\sum_{\lambda=1}^{\Lambda} H(X_\lambda)}{KNT} \tag{8.52}$$

$$\geq \min_{\boldsymbol{\mathcal{M}}} \frac{H(X_{[\Lambda]}|Y_{[\Lambda]^c})}{KNT} \tag{8.53}$$

$$\geq \min_{\boldsymbol{\mathcal{M}}} \sum_{\lambda\in[\Lambda]} \sum_{j=1}^{\Lambda} a^{(j,\lambda)}_{[\Lambda],\boldsymbol{\mathcal{M}}} \frac{\sum_{q\in[\Lambda]} L_q - j\cdot L_\lambda}{KNj^2}, \tag{8.54}$$

where (8.54) is due to Lemma 12.

Recalling that $\sum_{\lambda=1}^{\Lambda} \gamma_\lambda = t$, under the assumption of homogeneous file assignment we have that

$$a^{(j)}_{[\Lambda],\boldsymbol{\mathcal{M}}} = \begin{cases} N & j = t, \\ 0 & j \neq t, \end{cases} \tag{8.55}$$

and that

$$a^{(t,\lambda)}_{[\Lambda],\boldsymbol{\mathcal{M}}} = \gamma_\lambda N,\ \lambda \in [\Lambda]. \tag{8.56}$$

Applying this assumption in (8.54), we obtain that the optimal communication load $\tau^*_{hom}(\mathbf{L}, \boldsymbol{\gamma})$can be lower bounded as

$$\tau^*_{hom}(\mathbf{L}, \boldsymbol{\gamma}) \geq \min_{\boldsymbol{\mathcal{M}}} \sum_{\lambda\in[\Lambda]} a^{(t,\lambda)}_{[\Lambda],\boldsymbol{\mathcal{M}}} \frac{\sum_{q\in[\Lambda]} L_q - t\cdot L_\lambda}{KNt^2} \tag{8.57}$$

$$\overset{(a)}{=} \min_{\boldsymbol{\mathcal{M}}} \frac{a^{(t)}_{[\Lambda],\boldsymbol{\mathcal{M}}} \sum_{q\in[\Lambda]} L_q - \sum_{\lambda\in[\Lambda]} a^{(t,\lambda)}_{[\Lambda],\boldsymbol{\mathcal{M}}} \cdot L_\lambda}{KNt} \tag{8.58}$$

$$\overset{(b)}{=} \min_{\boldsymbol{\mathcal{M}}} \frac{\sum_{q\in[\Lambda]} L_q - \sum_{\lambda\in[\Lambda]} \gamma_\lambda L_\lambda}{Kt} \tag{8.59}$$

$$\overset{(c)}{=} \frac{\sum_{\lambda\in[\Lambda]} L_\lambda(1-\gamma_\lambda)}{Kt}, \tag{8.60}$$

where (a) follows from (8.49), (b) follows from (8.55) and (8.56), and (c) from the fact that $\frac{\sum_{\lambda\in[\Lambda]} L_\lambda(1-\gamma_\lambda)}{Kt}$ does not depend on the specific file assignment $\boldsymbol{\mathcal{M}}$. The proof of Theorem 11 is complete.

Chapter 9

Conclusions and Future Directions

In this chapter, we quickly revisit the major contributions of this thesis and present some possible future directions.

9.1 Shared-Cache Networks with Single-Antenna Transmitter

Motivated by realistic cache-aided heterogeneous networks, in this thesis we have studied the fundamental limits of shared-cache networks with (potentially) heterogeneously populated caches. While we mostly focused on the case where each user is associated to only one cache, for a specific symmetric network configuration with as many users as caches we have assumed that users are associated to different many helper caches.

9.1.1 Topology-Agnostic and Topology-Aware Scenarios

Brief Summary: For the topology-agnostic scenario, where the number of users connected to each cache is not known during the caching phase, we have characterized the optimal normalized delivery time under the assumption of uncoded cache placement. This result shows that, as expected, the sum-DoF of $t + 1 = \Lambda\gamma + 1$ is achievable only when users are distributed uniformly among the caches. This is in contrast to the topology-aware scenario where we have shown that a proper memory allocation as a function of the number of users associated to the caches allows for multicasting opportunities that, in turn, result in a sum-DoF of $t + 1$ regardless of the specific cache occupancy vector. At the same time, a carefully designed non-uniform memory allocation leads to higher local caching gains than the uniform case, thus surprisingly implying that the more skewed the cache occupancy vector is, the lower will be the NDT required to serve all the users. For this topology-aware scenario, the optimality of our achievable performance requires — in addition to the aforementioned uncoded cache placement assumption — also the assumption that all the bits of the library are stored across the caches the same number of times (homogeneous placement). Finally, we have demonstrated the benefits of memory allocation also in the case when only the average cache occupancies are known in the placement phase, and we have shown how even such partial knowledge far outperforms

the topology-agnostic scheme. We recall that all the above results also apply to the equivalent multiple file request problem.

Discussions for Possible Future Directions: Dealing with the heterogeneity of the setting raised interesting challenges in redesigning converse bounds as well as redesigning coded caching schemes, which generally thrive on uniformity. We hope and we believe that the tools developed for the study of these problems can be useful for other heterogeneous settings. For example, we have seen how the proposed memory allocation for the topology-aware scenario is strongly related to the one used in [38] for the dedicated-cache setting where the link rates between the transmitter and receivers are different. We hope that the techniques used in our developed bound for the topology-aware scenario could be exploited also in this unequal link rates setting in order to provide better information theoretic guarantees than the one presented in [38]. Furthermore, we observe that the high gains achieved by our topology-aware scheme come with a price of a high subpacketization requirement, which we recall takes the form $S = \sum_{\tau \in C_t^{[\Lambda]}} \prod_{j=1}^{t} L_{\tau(j)}$. In the finite file size regime with a large number of caches and users, this high subpacketization requirement might significantly limit the actual gains. Therefore, a possible interesting direction for future work could aim at designing achievable coded caching schemes with lower subpacketization requirements. Finally, we recall that our work assumes that the cost of fetching data from the caches is zero. However, while it is true that the data rates at which a cache-aided access point can serve the users are much higher than those at which a macro base station can serve them, it is also true that such rates are finite and therefore fetching content from the caches is in general a costly operation. This rises the need of studying shared-cache networks where the access to the caches has a non zero cost.

9.1.2 Multi-Access Shared-Cache Network

In this thesis we have considered a special type of multi-access shared-cache networks where each user is associated to z neighboring caches and where the number of caches equals the number of users. When our results were published, our work was the first to show that a global caching gain higher than the one achievable in the single access setting — where each user is associated to only one cache — is achievable along with the full local caching gain arising from the several caches to which each user is connected. We showed this interesting fact by proposing two achievable schemes for the special cases where $K\gamma = 2$ and $z = \frac{K-1}{K\gamma}$. For the latter case, our proposed scheme is optimal and it shows an ability to serve $K\gamma z + 1$ users simultaneously as if there where zK caches in the network and each user had exclusive access to z of them. We acknowledge the limitations of our achievable schemes which are designed for these 2 aforementioned special cases. Despite after the publication of our results, several other works have provided additional contributions on this topic, characterizing the fundamental limits of such networks still remains an open problem, both in terms of achievable schemes and converse bounds.

9.2 Cache-Aided MISO Broadcast Channel

Brief Summary: The first contributions of this thesis on the cache-aided MISO BC were obtained for the setting where each user is associated to a cache which, in turn, serves an arbitrary number of users. For this shared-cache network, we have assumed a topology-agnostic scenario and we have focused on the case where the number of antennas N_0 at the transmitter is smaller than the number of users per cache. For this described setting, we characterized the optimal normalized delivery time (DoF regime) under the assumption of uncoded cache placement, and we showed that the gain coming from having multiple antennas allows for a multiplicative reduction of N_0 in the NDT. For the complementary regime where N_0 can be higher than the number of users per cache, the optimal NDT remains generally unknown. However, for the special case where the users are distributed uniformly among the caches (with an integer constraint on the ratio between N_0 and the number of users per cache $\frac{K}{\Lambda}$) we have proposed a scheme achieving the optimal one-shot linear DoF of $K\gamma + N_0$ regardless of the value of Λ, as long as $\Lambda \geq \frac{K}{N_0}$. This result suggests that in a dedicated-cache setting where — motivated by subpacketization constraints — we use user grouping such that the users in the same group store the same content, arranging users in less than $\frac{K}{N_0}$ groups does not allow to achieve the optimal one-shot linear DoF of the network. Furthermore, for the dedicated-cache setting where the number of antennas N_0 at the transmitter exceeds the total cache redundancy $K\gamma$, we proposed a scheme that achieves a DoF of $K\gamma + \alpha$ ($\alpha \leq N_0$) with a low subpacketization requirement. We analysed the performance of this scheme also in the finite SNR regime where we have used optimized MMSE-type beamformers that require low computational complexity. Interestingly, the parameter α can be tuned both to control subpacketization and to trade multiplexing gains with beamforming gains, which we know being very important at low SNR regime. Finally, the low computational complexity of the beamforming design combined with the low subpacketization allows our scheme to be eligible for being used in large networks with many users and many antennas at the transmitter.

Discussions for Possible Future Directions: With minor modifications, our topology-agnostic multi-round scheme presented in Section 3.3 could be applied to the regime where $N_0 \geq L_\lambda, \forall \lambda \in [\Lambda]$. Also, we recall that our developed converse bound in Section 3.4 holds for this aforementioned regime. However, we know that, for this regime, the converse bound is loose, and we believe that the aforementioned multi-round scheme fails to achieve the optimal performance. Therefore, the optimal DoF of the topolgy-agnostic setting for this regime where $N_0 \geq L_\lambda, \forall \lambda \in [\Lambda]$ remains an open problem. In Chapter 4, we have briefly mentioned that the topology-aware scheme developed for the shared-link shared-cache network can be extended to the case where the transmitter is equipped with multiple antennas as long as $N_0 \leq L_\lambda, \forall \lambda \in [\Lambda]$. The details were not provided in this thesis. For the complementary regime where $N_0 \geq L_\lambda, \forall \lambda \in [\Lambda]$, we believe that if the optimal achievable scheme for the topology-agnostic scenario would be known, then, in light of the results of this thesis, the extension of such scheme to the topology-aware scenario might be straightforward. Finally, for the dedicated-cache setting we believe

that an interesting future direction would be the one of working towards a decentralized variation of the proposed scheme in Chapter 6, similarly to [53] where each user picks at random one of some pre-defined cache states. We believe that such a practical ramification of the proposed scheme could still achieve good finite SNR performance.

9.3 Cache-aided Networks with Heterogeneous QoS requirements

Brief Summary: In this thesis, we also characterized the rate-memory tradeoff for the coded caching problem with multi-layer coded files for the case when, in the uncoded cache placement, the server knows only the number of users requiring a given QoS level but does not know the specific QoS requirement of each user. To this end, we developed an information theoretic converse which in turn defined the design of an optimal scheme by defining how much of each layer should be placed in the caches. This interesting back-and-forth between the converse and the scheme, nicely highlights the usefulness of finding exact information theoretic bounds, since such exact bounds may have the potential to recreate the structure of the optimal scheme.

Discussions for Possible Future Directions: With the aim of characterizing the fundamental limits of networks with heterogeneous QoS requirements, studying the case when, in the cache placement phase, the server is aware of the QoS requirement of each user could be an interesting extension. This latter case has been already investigated in the literature, but the exact optimal memory-rate tradeoff is still unknown. If we allowed the cache size of each user to be optimized as a function of its requested QoS level and potentially the ones of the other users, we believe that the ideas behind the topology-aware scheme described in Section 4.4 for the shared-cache networks, could be applied for this coded caching problem where the QoS requirements of each user are known during the caching phase. At the same time, we believe that the tools used in the lower bound developed in Section 4.5 may also be exploited and applied to the aforementioned problem.

9.4 Heterogeneous Coded Distributed Computing

Brief Summary: The last problem that we addressed in this thesis is the coded distributed computing problem with heterogeneous computational powers. For a MapReduce-like framework with given reduce load vector $\mathbf{L}$ and a total computation load t, we have provided a file assignment and shuffle scheme which we proved to be order-optimal under the two reasonable assumptions of homogeneous file assignment and one-shot shuffling. Similarly as it happened for the topology-aware heterogeneous shared-cache problem, the order-optimality is obtained with a file assignment strategy that assigns more files of the dataset, i.e. more computation load, to those nodes that will have to compute more output functions in the reduce phase. Indeed, the proposed scheme can be seen as a D2D version of the scheme presented in Section 4.4. Furthermore, we have also seen that in order to minimize the reduce time, as expected, one should assign to each computing node

a number of output functions that is proportional to its computational power. Finally, we also provided a lower bound on the communication load for an arbitrary output function assignment and an arbitrary allocation of the total computation load across the nodes.

Discussions for Possible Future Directions: Part of our future effort on this topic will be to extend the shuffle scheme proposed in this thesis to the case with arbitrary computation load and arbitrary reduce load at each node, for which we have already provided a converse bound. For this latter problem, some recent advances have been recently published (see the related works subsection in 8.1.1). Finally, we believe that — while we mostly focused on minimizing the shuffle time and, separately, the reduce time — one should consider the much harder problem of identifying the optimal file assignment, shuffle scheme and output function assignment that jointly minimize the sum of the map, shuffle and reduce time for a MapReduce-like distributed computing system with heterogenenous computational powers.

Appendices

Appendix A

Appendix of Chapter 3

A.1 An Illustrative Example for the Converse

We here give an example of deriving the converse for Theorem 2, emphasizing on how to convert the caching problem to the index-coding problem, and how to choose acyclic subgraphs. We consider the case of having $K = 9$ receiving users, and a transmitter with $N_0 = 2$ transmit antennas having access to a library of $N = 9$ files of unit size. We also assume that there are $\Lambda = 3$ caching nodes, of total normalized cache size $t = \Lambda\gamma$. We will focus on deriving the bound for the cache occupancy vector $\mathbf{L} = (4, 3, 2)$, meaning that we are interested in the setting where one cache is associated to 4 users, one cache to 3 users and one cache associated to 2 users.

Each file $W^{(n)}$ is split into $2^\Lambda = 8$ disjoint subfiles $W^{(i)}_{\mathcal{T}}, \mathcal{T} \in 2^{[3]}$ where each $\mathcal{T}$ describes the set of helper nodes in which $W^{(i)}_{\mathcal{T}}$ is cached. For instance, $W^{(1)}_{13}$ refers to the part of file $W^{(1)}$ that is stored in the first and third caching nodes.

As a first step, we present the construction of the set $\mathcal{D}_{\mathbf{L}}$. To this end, let us start by considering the demand $\mathbf{d} = (1, 2, 3, 4, 5, 6, 7, 8, 9)$ and one of the 6 permutations $\pi \in S_3$; for example, let us start by considering $\pi(1) = 2, \pi(2) = 3, \pi(3) = 1$. Toward reordering $\mathbf{d}$ to reflect $\mathbf{L}$, we construct

$$\mathbf{d}^{'}_1 = (1, 2, 3, 4), \quad \mathbf{d}^{'}_2 = (5, 6, 7), \quad \mathbf{d}^{'}_3 = (8, 9)$$

to obtain the reordered demand vector

$$\begin{aligned}\mathbf{d}(\mathcal{U}) &= (\mathbf{d}^{'}_{\pi^{-1}(1)}, \mathbf{d}^{'}_{\pi^{-1}(2)}, \mathbf{d}^{'}_{\pi^{-1}(3)}) \\ &= (\mathbf{d}^{'}_3, \mathbf{d}^{'}_1, \mathbf{d}^{'}_2)\end{aligned}$$

which in turn yields $\mathbf{d}_1 = (8, 9), \mathbf{d}_2 = (1, 2, 3, 4), \mathbf{d}_3 = (5, 6, 7)$. Similarly, we can construct the remaining 5 demands $\mathbf{d}(\mathcal{U})$ associated to the other 5 permutations $\pi \in S_3$. Finally, the procedure is repeated for all other worst-case demand vectors. These vectors are part of set $\mathcal{D}_{\mathbf{L}}$.

With the users demands $\mathbf{d}(\mathcal{U})$ known to the server, the delivery problem is translated into an index coding problem with a side information graph of $K2^{\Lambda-1} = 9 \cdot 2^2$ nodes. For

each requested file $W^{(\mathbf{d}_\lambda(j))}$, we write down the 4 subfiles that the requesting user does not have in its assigned cache. Hence, a given user of the caching problem requiring 4 subfiles from the main server, is replaced by 4 different new users in the index coding problem. Each of these users request a different subfile and are connected to the same cache λ as the original user. The nodes of the 6 side-information graphs corresponding to the aforementioned vectors $\mathbf{d}(\mathcal{U})$ (one for each permutation $\pi \in S_3$) for demand $\mathbf{d} = (1, 2, 3, 4, 5, 6, 7, 8, 9)$, are depicted in Figure A.1.

For each side-information graph, we develop a lower bound as in Lemma 1. We recall that the lemma applies to acyclic subgraphs, which we create as follows; for each permutation[1] $\sigma \in S_3$, a set of nodes forming an acyclic subgraph is

$$\begin{aligned}
&\{W_{\mathcal{T}_1}^{(\mathbf{d}_{\sigma(1)}(j))}\}_{j=1}^{|\mathcal{U}_{\sigma(1)}|} \text{ for all } \mathcal{T}_1 \subseteq \{1,2,3\}\backslash\{\sigma(1)\},\\
&\{W_{\mathcal{T}_2}^{(\mathbf{d}_{\sigma(2)}(j))}\}_{j=1}^{|\mathcal{U}_{\sigma(2)}|} \text{ for all } \mathcal{T}_2 \subseteq \{1,2,3\}\backslash\{\sigma(1),\sigma(2)\},\\
&\{W_{\mathcal{T}_3}^{(\mathbf{d}_{\sigma(3)}(j))}\}_{j=1}^{|\mathcal{U}_{\sigma(3)}|} \text{ for all } \mathcal{T}_3 \subseteq \{1,2,3\}\backslash\{\sigma(1),\sigma(2),\sigma(3)\}.
\end{aligned}$$

Based on this construction of acyclic graphs, our task now is to choose a permutation $\sigma_s \in S_3$ that forms the maximum-sized acyclic subgraph. For the case where $\mathbf{d}_1 = (8, 9), \mathbf{d}_2 = (1, 2, 3, 4)$ and $\mathbf{d}_3 = (5, 6, 7)$, it can be easily verified that such a permutation σ_s is the one with $\sigma_s(1) = 2$,$\sigma_s(2) = 3$ and $\sigma_s(3) = 1$. In Figure A.1, for each of the six graphs, we underline the nodes corresponding to the acyclic subgraph that is formed by such permutation σ_s. The outer bound now involves adding the sizes of these chosen (underlined) nodes. For example, for the demand $\mathbf{d}(\mathcal{U}) = ((8, 9), (1, 2, 3, 4), (5, 6, 7))$ (this corresponds to the lower center graph), the lower bound in (3.42) becomes

$$\begin{aligned}
T^*(\mathbf{d}(\mathcal{U}), \mathbf{Z}) \geq \frac{1}{2}(&|W_\emptyset^{(1)}| + |W_1^{(1)}| + |W_3^{(1)}| + |W_{13}^{(1)}| + |W_\emptyset^{(2)}|\\
&+ |W_1^{(2)}| + |W_3^{(2)}| + |W_{13}^{(2)}| + |W_\emptyset^{(3)}| + |W_1^{(3)}|\\
&+ |W_3^{(3)}| + |W_{13}^{(3)}| + |W_\emptyset^{(4)}| + |W_1^{(4)}| + |W_3^{(4)}|\\
&+ |W_{13}^{(4)}| + |W_\emptyset^{(5)}| + |W_1^{(5)}| + |W_\emptyset^{(6)}| + |W_1^{(6)}|\\
&+ |W_\emptyset^{(7)}| + |W_1^{(7)}| + |W_\emptyset^{(8)}| + |W_0^{(9)}|). \qquad \text{(A.1)}
\end{aligned}$$

The lower bounds for the remaining 5 vectors $\mathbf{d}(\mathcal{U})$ for the same $\mathbf{d} = (1, 2, 3, 4, 5, 6, 7, 8, 9)$, are given in a similar way, again by adding the (underlined) nodes of the corresponding acyclic subgraphs (again see Figure A.1).

Subsequently, the procedure is repeated for all $P(N, K) = K! = 9!$ worst-case demand vectors $\mathbf{d} \in \mathcal{D}_{wc}$. Finally, all the $P(N, K) \cdot \Lambda! = 9! \cdot 3!$ bounds are averaged to get

$$\begin{aligned}
&T(\mathbf{L}, \mathbf{Z}) \geq\\
&\frac{1}{2}\frac{1}{9! \cdot 3!} \sum_{\mathbf{d}(\mathcal{U}) \in \mathcal{D}_\mathbf{L}} \sum_{\lambda \in [3]} \sum_{j=1}^{L_{\sigma_s(\lambda)}} \sum_{\mathcal{T}_\lambda \subseteq [3]\backslash\{\sigma_s(1),\ldots,\sigma_s(\lambda)\}} |W_{\mathcal{T}_\lambda}^{\mathbf{d}_{\boldsymbol{\sigma_s}(\boldsymbol{\lambda})}(j)}| \qquad \text{(A.2)}
\end{aligned}$$

[1] We caution the reader not to confuse the current permutations (σ) that are used to construct large-sized acyclic graphs, with the aforementioned permutations π which are used to construct $\mathcal{D}_\mathbf{L}$.

$\mathbf{d}_1 = (1,2,3,4), \mathbf{d}_2 = (5,6,7), \mathbf{d}_3 = (8,9)$

$$\begin{array}{cccc} \underline{W_{\emptyset}^{(1)}} & \underline{W_{2}^{(1)}} & \underline{W_{3}^{(1)}} & \underline{W_{23}^{(1)}} \\ \underline{W_{\emptyset}^{(2)}} & \underline{W_{2}^{(2)}} & \underline{W_{3}^{(2)}} & \underline{W_{23}^{(2)}} \\ \underline{W_{\emptyset}^{(3)}} & \underline{W_{2}^{(3)}} & \underline{W_{3}^{(3)}} & \underline{W_{23}^{(3)}} \\ \underline{W_{\emptyset}^{(4)}} & \underline{W_{2}^{(4)}} & \underline{W_{3}^{(4)}} & \underline{W_{23}^{(4)}} \\ \underline{W_{\emptyset}^{(5)}} & W_{1}^{(5)} & \underline{W_{3}^{(5)}} & W_{13}^{(5)} \\ \underline{W_{\emptyset}^{(6)}} & W_{1}^{(6)} & \underline{W_{3}^{(6)}} & W_{13}^{(6)} \\ \underline{W_{\emptyset}^{(7)}} & W_{1}^{(7)} & \underline{W_{3}^{(7)}} & W_{13}^{(7)} \\ \underline{W_{\emptyset}^{(8)}} & W_{1}^{(8)} & W_{2}^{(8)} & W_{12}^{(8)} \\ \underline{W_{\emptyset}^{(9)}} & W_{1}^{(9)} & W_{2}^{(9)} & W_{12}^{(9)} \end{array}$$

$\mathbf{d}_1 = (1,2,3,4), \mathbf{d}_2 = (8,9), \mathbf{d}_3 = (5,6,7)$

$$\begin{array}{cccc} \underline{W_{\emptyset}^{(1)}} & \underline{W_{2}^{(1)}} & \underline{W_{3}^{(1)}} & \underline{W_{23}^{(1)}} \\ \underline{W_{\emptyset}^{(2)}} & \underline{W_{2}^{(2)}} & \underline{W_{3}^{(2)}} & \underline{W_{23}^{(2)}} \\ \underline{W_{\emptyset}^{(3)}} & \underline{W_{2}^{(3)}} & \underline{W_{3}^{(3)}} & \underline{W_{23}^{(3)}} \\ \underline{W_{\emptyset}^{(4)}} & \underline{W_{2}^{(4)}} & \underline{W_{3}^{(4)}} & \underline{W_{23}^{(4)}} \\ \underline{W_{\emptyset}^{(5)}} & W_{1}^{(5)} & \underline{W_{2}^{(5)}} & W_{12}^{(5)} \\ \underline{W_{\emptyset}^{(6)}} & W_{1}^{(6)} & \underline{W_{2}^{(6)}} & W_{12}^{(6)} \\ \underline{W_{\emptyset}^{(7)}} & W_{1}^{(7)} & \underline{W_{2}^{(7)}} & W_{12}^{(7)} \\ \underline{W_{\emptyset}^{(8)}} & W_{1}^{(8)} & W_{3}^{(8)} & W_{13}^{(8)} \\ \underline{W_{\emptyset}^{(9)}} & W_{1}^{(9)} & W_{3}^{(9)} & W_{13}^{(9)} \end{array}$$

$\mathbf{d}_1 = (5,6,7), \mathbf{d}_2 = (1,2,3,4), \mathbf{d}_3 = (8,9)$

$$\begin{array}{cccc} \underline{W_{\emptyset}^{(1)}} & \underline{W_{1}^{(1)}} & \underline{W_{3}^{(1)}} & \underline{W_{13}^{(1)}} \\ \underline{W_{\emptyset}^{(2)}} & \underline{W_{1}^{(2)}} & W_{3}^{(1)} & \underline{W_{13}^{(2)}} \\ \underline{W_{\emptyset}^{(3)}} & \underline{W_{1}^{(3)}} & \underline{W_{3}^{(3)}} & \underline{W_{13}^{(3)}} \\ \underline{W_{\emptyset}^{(4)}} & \underline{W_{1}^{(4)}} & \underline{W_{3}^{(4)}} & \underline{W_{13}^{(4)}} \\ \underline{W_{\emptyset}^{(5)}} & W_{2}^{(5)} & \underline{W_{3}^{(5)}} & W_{23}^{(5)} \\ \underline{W_{\emptyset}^{(6)}} & W_{2}^{(6)} & \underline{W_{3}^{(6)}} & W_{23}^{(6)} \\ \underline{W_{\emptyset}^{(7)}} & W_{2}^{(7)} & \underline{W_{3}^{(7)}} & W_{23}^{(7)} \\ \underline{W_{\emptyset}^{(8)}} & W_{1}^{(8)} & W_{2}^{(8)} & W_{12}^{(8)} \\ \underline{W_{\emptyset}^{(9)}} & W_{1}^{(9)} & W_{2}^{(9)} & W_{12}^{(9)} \end{array}$$

$\mathbf{d}_1 = (5,6,7), \mathbf{d}_2 = (8,9), \mathbf{d}_3 = (1,2,3,4)$

$$\begin{array}{cccc} \underline{W_{\emptyset}^{(1)}} & \underline{W_{1}^{(1)}} & \underline{W_{2}^{(1)}} & \underline{W_{12}^{(1)}} \\ \underline{W_{\emptyset}^{(2)}} & \underline{W_{1}^{(2)}} & \underline{W_{2}^{(2)}} & \underline{W_{12}^{(2)}} \\ \underline{W_{\emptyset}^{(3)}} & \underline{W_{1}^{(3)}} & \underline{W_{2}^{(3)}} & \underline{W_{12}^{(3)}} \\ \underline{W_{\emptyset}^{(4)}} & \underline{W_{1}^{(4)}} & \underline{W_{2}^{(4)}} & \underline{W_{12}^{(4)}} \\ \underline{W_{\emptyset}^{(5)}} & \underline{W_{2}^{(5)}} & W_{3}^{(5)} & W_{23}^{(5)} \\ \underline{W_{\emptyset}^{(6)}} & \underline{W_{2}^{(6)}} & W_{3}^{(6)} & W_{23}^{(6)} \\ \underline{W_{\emptyset}^{(7)}} & \underline{W_{2}^{(7)}} & W_{3}^{(7)} & W_{23}^{(7)} \\ \underline{W_{\emptyset}^{(8)}} & W_{1}^{(8)} & W_{3}^{(8)} & W_{13}^{(8)} \\ \underline{W_{\emptyset}^{(9)}} & W_{1}^{(9)} & W_{3}^{(9)} & W_{13}^{(9)} \end{array}$$

$\mathbf{d}_1 = (8,9), \mathbf{d}_2 = (1,2,3,4), \mathbf{d}_3 = (5,6,7)$

$$\begin{array}{cccc} \underline{W_{\emptyset}^{(1)}} & \underline{W_{1}^{(1)}} & \underline{W_{3}^{(1)}} & \underline{W_{13}^{(1)}} \\ \underline{W_{\emptyset}^{(2)}} & \underline{W_{1}^{(2)}} & \underline{W_{3}^{(2)}} & \underline{W_{13}^{(2)}} \\ \underline{W_{\emptyset}^{(3)}} & \underline{W_{1}^{(3)}} & \underline{W_{3}^{(3)}} & \underline{W_{13}^{(3)}} \\ \underline{W_{\emptyset}^{(4)}} & \underline{W_{1}^{(4)}} & \underline{W_{3}^{(4)}} & \underline{W_{13}^{(4)}} \\ \underline{W_{\emptyset}^{(5)}} & \underline{W_{1}^{(5)}} & W_{2}^{(5)} & W_{12}^{(5)} \\ \underline{W_{\emptyset}^{(6)}} & \underline{W_{1}^{(6)}} & W_{2}^{(6)} & W_{12}^{(6)} \\ \underline{W_{\emptyset}^{(7)}} & \underline{W_{1}^{(7)}} & W_{2}^{(7)} & W_{12}^{(7)} \\ \underline{W_{\emptyset}^{(8)}} & W_{2}^{(8)} & W_{3}^{(8)} & W_{23}^{(8)} \\ \underline{W_{\emptyset}^{(9)}} & W_{2}^{(9)} & W_{3}^{(9)} & W_{23}^{(9)} \end{array}$$

$\mathbf{d}_1 = (8,9), \mathbf{d}_2 = (5,6,7), \mathbf{d}_3 = (1,2,3,4)$

$$\begin{array}{cccc} \underline{W_{\emptyset}^{(1)}} & \underline{W_{1}^{(1)}} & \underline{W_{2}^{(1)}} & \underline{W_{12}^{(1)}} \\ \underline{W_{\emptyset}^{(2)}} & \underline{W_{1}^{(2)}} & \underline{W_{2}^{(2)}} & \underline{W_{12}^{(2)}} \\ \underline{W_{\emptyset}^{(3)}} & \underline{W_{1}^{(3)}} & \underline{W_{2}^{(3)}} & \underline{W_{12}^{(3)}} \\ \underline{W_{\emptyset}^{(4)}} & \underline{W_{1}^{(4)}} & \underline{W_{2}^{(4)}} & \underline{W_{12}^{(4)}} \\ \underline{W_{\emptyset}^{(5)}} & \underline{W_{1}^{(5)}} & W_{3}^{(5)} & W_{13}^{(5)} \\ \underline{W_{\emptyset}^{(6)}} & \underline{W_{1}^{(6)}} & W_{3}^{(6)} & W_{13}^{(6)} \\ \underline{W_{\emptyset}^{(7)}} & \underline{W_{1}^{(7)}} & W_{3}^{(7)} & W_{13}^{(7)} \\ \underline{W_{\emptyset}^{(8)}} & W_{2}^{(8)} & W_{3}^{(8)} & W_{23}^{(8)} \\ \underline{W_{\emptyset}^{(9)}} & W_{2}^{(9)} & W_{3}^{(9)} & W_{23}^{(9)} \end{array}$$

Figure A.1 – Nodes of the side information graphs corresponding to demand vector $\mathbf{d} = (1,2,3,4,5,6,7,8,9)$ and cache occupancy vector $\mathbf{L} = (4,3,2)$.

which is rewritten as

$$T(\boldsymbol{\mathcal{L}}, \mathbf{Z}) \geq$$
$$\frac{1}{2}\frac{1}{9! \cdot 3!} \sum_{i=0}^{3} \sum_{n\in[9]} \sum_{\mathcal{T}\subseteq[3]:|\mathcal{T}|=i} |W_{\mathcal{T}}^{(n)}| \cdot \underbrace{\sum_{\mathbf{d}(\mathcal{U})\in\mathcal{D}_{\mathbf{L}}} \mathbb{1}_{\mathcal{V}_{\mathcal{J}_s^{\mathbf{d}(\mathcal{U})}}}(W_{\mathcal{T}}^{(n)})}_{Q_i(W_{\mathcal{T}}^{(n)})} . \tag{A.3}$$

After the evaluation of the term $Q_i(W_{\mathcal{T}}^{(n)})$, the bound in (A.3) can be written in a more compact form as

$$T(\mathbf{L}, \mathbf{Z}) \geq \frac{1}{2} \sum_{i=0}^{3} \frac{\sum_{r=1}^{3-i} L_r \binom{3-r}{i}}{9\binom{3}{i}} x_i \tag{A.4}$$
$$\geq \mathrm{Conv}_{i\in[\Lambda]_0} \left(\frac{1}{2} \frac{\sum_{r=1}^{3-i} L_r \binom{3-r}{i}}{\binom{3}{i}} \right) \tag{A.5}$$

where the proof of the transition from (A.3) to (A.4) and from (A.4) to (A.5) can be found in the general proof (Section 3.4).

A.2 Collection of Proofs

A.2.1 Proof of Lemma 1

In the addressed problem, we consider a MISO broadcast channel with N_0 antennas at the transmitter serving K receivers with some side information due to caches. In the wired setting this (high-SNR setting) is equivalent to the distributed index coding problem with N_0 senders $J_1, \ldots, J_{N_0}$, all having knowledge of the entire set of messages, and each being connected via an (independent) broadcast line link of capacity $C_{J_i} = 1, i \in [N_0]$ to the K receivers which hold side information. This multi-sender index coding problem is addressed in [54]. By adapting the achievable rate result in [54, Corollary1] to our problem, we get

$$\sum_{\mathcal{V}\in\mathcal{V}_{\mathcal{J}}} R_{\mathcal{V}} \leq \sum_{i\in[N_0]} C_{J_i} \tag{A.6}$$

($R_{\mathcal{V}} = \frac{|\mathcal{V}|}{T}$ is the rate for message $\mathcal{V}$), that yields

$$\sum_{\mathcal{V}\in\mathcal{V}_{\mathcal{J}}} \frac{|\mathcal{V}|}{T} \leq N_0 \tag{A.7}$$

which, when inverted, gives the bound in Lemma 1. □

A.2.2 Proof of Lemma 2

Consider a permutation σ where the subfiles $W_{\mathcal{T}_\lambda}^{(\mathbf{d}_{\sigma(\lambda)}(j))}, \forall j \in \mathcal{U}_{\sigma(\lambda)}$ for all $\mathcal{T}_\lambda \subseteq [\Lambda] \setminus \{\sigma(1), \ldots, \sigma(\lambda)\}$ are all placed in row λ of a matrix whose rows are labeled by $\lambda = 1, 2, \ldots, \Lambda$. The index coding users corresponding to subfiles in row λ only know (as side information) subfiles $W_{\mathcal{T}}^{(d_k)}$, $\mathcal{T} \ni \sigma(\lambda)$. Consequently each user/node of row λ does not know any of the subfiles in the same row[2] nor in the previous rows. As a result, the proposed set of subfiles chosen according to permutation σ, forms a subgraph that does not contain any cycle.

A basic counting argument can tell us that the number of subfiles — in the acyclic subgraph formed by any permutation $\sigma \in S_\Lambda$ — that are stored in exactly i caches, is

$$\sum_{r=1}^{\Lambda-i} |\mathcal{U}_{\sigma(r)}| \binom{\Lambda - r}{i}. \tag{A.8}$$

This means that the total number of subfiles in the acyclic subgraph is simply

$$\sum_{i=0}^{\Lambda} \sum_{r=1}^{\Lambda-i} |\mathcal{U}_{\sigma(r)}| \binom{\Lambda - r}{i}. \tag{A.9}$$

This number is maximized when the permutation σ guarantees that the vector $(|\mathcal{U}_{\sigma(1)}|, |\mathcal{U}_{\sigma(2)}|, \ldots, |\mathcal{U}_{\sigma(\Lambda)}|)$ is in descending order. This maximization is achieved with our choice of the ordering permutation σ_s (as this was defined in the notation part) when constructing the acyclic graphs. □

A.2.3 Proof of Equation (3.52)

Here, through a combinatorial argument, we derive $Q_i(W_{\mathcal{T}}^{(n)})$, that is the number of times that a subfile $W_{\mathcal{T}}^{(n)}$ with index size $|\mathcal{T}| = i$ appears in all the acyclic subgraphs chosen to develop the lower bound.

There are $\binom{N-1}{K-1}$ subsets $\Upsilon_m, m \in [\binom{N-1}{K-1}]$ out of $\binom{N}{K}$ unordered subsets of K files from the set $\{W^{(j)}, j \in [N]\}$ that contain file $W^{(n)}$, and for each Υ_m there exists $K!$ different demand vectors $\mathbf{d}'$. For each Υ_m, among all possible demand vectors, a subfile $W_{\mathcal{T}}^{(n)} : |\mathcal{T}| = i$ appears in the side information graph an equal number of times. For a fixed Υ_m, file $W^{(n)}$ is requested by a user connected to any helper node with a certain cardinality L_r. By construction, $Q_i(W_{\mathcal{T}}^{(n)})$ can be rewritten as

$$\begin{aligned} Q_i(W_{\mathcal{T}}^{(n)}) &= \sum_{\mathbf{d} \in \mathcal{D}_{wc}} \sum_{\pi \in S_\Lambda} \mathbb{1}_{\mathcal{V}_{\mathcal{J}_s^{\mathbf{d}_r(\mathcal{U})}}}(W_{\mathcal{T}}^{(n)}) \\ &= \binom{N-1}{K-1} \sum_{r=1}^{\Lambda} \sum_{\mathbf{d}'_r \in \mathcal{D}_{wc}} \sum_{\pi \in S_\Lambda} \mathbb{1}_{\mathcal{V}_{\mathcal{J}_s^{\mathbf{d}'_r(\mathcal{U})}}}(W_{\mathcal{T}}^{(n)}) \end{aligned} \tag{A.10}$$

[2]Notice that the index coding users/nodes who are associated to the same cache, are not linked by any edge in the corresponding graph.

where $\mathbf{d}'_r$ denotes the subset of all demand vectors from Υ_m such that $n \in \mathbf{d}_\lambda : |\mathbf{d}_\lambda| = L_r$. The number of chosen maximum acyclic subgraphs containing $W_{\mathcal{T}}^{(n)}$ that arise from all the demand vectors $\mathbf{d}'_r(\mathcal{U})$ is evaluated as follows. After fixing the demands such that $n \in \mathbf{d}_\lambda : |\mathbf{d}_\lambda| = L_r$, then $W_{\mathcal{T}}^{(n)}$ appears in the side information graph only if it is requested by a user connected to helper node λ such that $\lambda \notin \mathcal{T}$, which corresponds to $(\Lambda - i)$ different *available* positions in the demand vector $\mathbf{d}'_r(\mathcal{U})$, since $|\mathcal{T}| = i$. After fixing one of the $(\Lambda - i)$ positions occupied by $\mathbf{d}_\lambda : |\mathbf{d}_\lambda| = L_r$, for the remaining demands $\mathbf{d}_\lambda : |\mathbf{d}_\lambda| = L_j, \forall j \in [\Lambda] \setminus \{r\}$ there are $P(\Lambda - i - 1, r - 1) \cdot (\Lambda - r)!$ possible ways to be placed into $\mathbf{d}$. After fixing the order of $\mathbf{d}_\lambda, \forall \lambda \in [\Lambda]$ in $\mathbf{d}$ and $n \in \mathbf{d}_\lambda : |\mathbf{d}_\lambda| = L_r$, there are L_r different positions in which n can be placed in $\mathbf{d}_\lambda : |\mathbf{d}_\lambda| = L_r$. This leaves out $L_r - 1$ positions with $K - 1$ different numbers from the considered set $\Upsilon_m \setminus \{n\}$, and the remaining $K - L_r$ positions in $\mathbf{d}'_r$ are filled with $K - L_r$ numbers. Therefore, there exist $L_r P(K-1, L_r - 1)(K - L_r)!$ different demand vectors where the subfile $W_{\mathcal{T}}^{(n)}$ will appear in the associated maximum acyclic subgraphs. Hence, the above jointly tell us that

$$\begin{aligned} Q_i(W_{\mathcal{T}}^{(n)}) = \binom{N-1}{K-1} \sum_{r=1}^{\Lambda} P(\Lambda - i - 1, r - 1) \\ \times (\Lambda - r)! L_r P(K-1, L_r - 1)(K - L_r)!(\Lambda - i) \end{aligned} \tag{A.11}$$

which concludes the proof. □

A.2.4 Transition from Equation (3.55) to (3.56)

The coefficient of x_i in equation (3.55), can be further simplified as follows

$$\begin{aligned} &\frac{Q_i}{\Lambda! P(N,K)} = \\ &= \frac{(N-1)!(N-K)!}{(K-1)!(N-K)!\Lambda!N!} \sum_{r=1}^{\Lambda} L_r P(K-1, L_r - 1) \\ &\quad \times (K - L_r)!(\Lambda - i) P(\Lambda - i - 1, r - 1)(\Lambda - r)! \\ &= \frac{1}{(K-1)!\Lambda!N} \\ &\quad \times \sum_{r=1}^{\Lambda} L_r \frac{(K-1)!(K-L_r)!(\Lambda - i)(\Lambda - i - 1)!(\Lambda - r)!}{(K - L_r)!(\Lambda - i - r)!} \\ &= \frac{1}{\Lambda! N} \sum_{r=1}^{\Lambda} L_r \frac{(K-1)!(\Lambda - i)!(\Lambda - r)!}{(K-1)!(\Lambda - i - r)!} \\ &= \frac{1}{N} \sum_{r=1}^{\Lambda} L_{\pi_s(r)} \frac{(\Lambda - i)!(\Lambda - r)! i!}{\Lambda!(\Lambda - i - r)! i!} \\ &= \frac{1}{N} \sum_{r=1}^{\Lambda} L_r \frac{\binom{\Lambda - r}{i}}{\binom{\Lambda}{i}} \end{aligned} \tag{A.12}$$

which concludes the proof. □

A.2.5 Monotonicity of $\{c_i\}$

Let us define the following sequences

$$(a_n)_{n\in[\Lambda-i]} \triangleq \left\{ \frac{\binom{\Lambda-n}{i}}{\binom{\Lambda}{i}}, n \in [\Lambda - i] \right\} \tag{A.13}$$

$$(b_n)_{n\in[\Lambda-i-1]} \triangleq \left\{ \frac{\binom{\Lambda-n}{i+1}}{\binom{\Lambda}{i+1}}, n \in [\Lambda - i - 1] \right\}. \tag{A.14}$$

It is easy to verify that $a_n \geq b_n$, $\forall n \in [\Lambda - i]$. Consider now the set of scalar numbers $\{V_j, j \in [\Lambda - i], V_j \in \mathbb{N}\}$. The inequality $a_n^* \geq b_n^*$, $\forall n \in [\Lambda - i]$ holds for

$$(a_n^*)_{n\in[\Lambda-i]} \triangleq \big\{ V_n \cdot a_n, n \in [\Lambda - i] \big\} \tag{A.15}$$

$$(b_n^*)_{n\in[\Lambda-i-1]} \triangleq \big\{ V_n \cdot b_n, n \in [\Lambda - i - 1] \big\}. \tag{A.16}$$

As a result, we have

$$\sum_{n\in[\Lambda-i]} V_n \cdot a_n \geq \sum_{n\in[\Lambda-i]} V_n \cdot b_n \tag{A.17}$$

which proves that $c_i \geq c_{i+1}$. □

A.2.6 Proof of (3.59)

Through the respective change of variables $t \triangleq \Lambda \frac{M}{N}$, $x_i' \triangleq \frac{x_i}{N}$ and $c_i' \triangleq \frac{c_i}{N_0}$, in equations (3.56), (3.54) and (3.58), we obtain

$$T(\mathbf{L}, \mathbf{Z}) \geq \sum_{i=0}^{\Lambda} x_i' c_i' \tag{A.18}$$

$$\sum_{i=0}^{\Lambda} x_i' = 1 \tag{A.19}$$

$$\sum_{i=0}^{\Lambda} i x_i' \leq t. \tag{A.20}$$

Let X denote a discrete integer-valued random variable with probability mass function $f_X(x) = \{x_i' \text{ if } x = i, \forall i \in \{0, 1, \ldots, \Lambda\}\}$, where the x_i' are those that satisfy equation (A.19). The value c_i' can also be seen as the realization of a random variable $Y \triangleq g(X)$, where $g(x) = \frac{\sum_{r=1}^{\Lambda-x} L_r \binom{\Lambda-r}{x}}{N_0 \binom{\Lambda}{x}}$, having the same probability mass function as X, i.e. $f_Y(y) = \{x_i' \text{ if } y = c_i', \forall i \in \{0, 1, \ldots, \Lambda\}\}$. Due to the equation in (A.20), the expectation of X is bounded as $\mathbb{E}[X] \leq t$. Similarly, (A.18) is equivalent to $T(\mathbf{L}, \mathbf{Z}) \geq \mathbb{E}[Y]$. From Jensen's inequality, we have $T(\mathbf{L}, \mathbf{Z}) \geq \mathbb{E}[Y] \geq g(\mathbb{E}[X])$. Next, we notice that function $g(x)$ is

monotonically decreasing because it is obtained from the decreasing sequence $\{c_i'\}$ by replacing integer i with a continuous real variable x. This allows for the lower bound

$$T(\mathbf{L},\mathbf{Z}) \geq g(\mathbb{E}[X]) \geq g(t) \overset{(a)}{=} \frac{\sum_{r=1}^{\Lambda-t} L_r \binom{\Lambda-r}{t}}{N_0 \binom{\Lambda}{t}} \tag{A.21}$$

where (a) is due the decreasing monotonicity of $g(x)$ and the fact that $\mathbb{E}[X] \leq t$. This concludes the proof. □

A.2.7 Proof of Equation (3.19)

We remind the reader that (for brevity of exposition, and without loss of generality) this part assumes that the $|\mathcal{U}_\lambda|$ are in decreasing order.

We define the following quantity

$$b_\lambda \triangleq |\mathcal{U}_1| - |\mathcal{U}_\lambda|$$

and rewrite the total number of transmissions using the above definition as

$$\begin{aligned}
&\sum_{j=1}^{|\mathcal{U}_1|} \binom{\Lambda}{\Lambda\gamma+1} - \binom{a_j}{\Lambda\gamma+1} \\
&= |\mathcal{U}_1| \binom{\Lambda}{\Lambda\gamma+1} - \sum_{j=1}^{|\mathcal{U}_1|} \binom{a_j}{\Lambda\gamma+1} \\
&= \sum_{i=1}^{\Lambda-\Lambda\gamma} (|\mathcal{U}_i| + b_i) \binom{\Lambda-i}{\Lambda\gamma} - \sum_{j=1}^{|\mathcal{U}_1|} \sum_{i=\Lambda\gamma}^{a_j-1} \binom{i}{\Lambda\gamma} \\
&= \sum_{i=1}^{\Lambda-\Lambda\gamma} |\mathcal{U}_i| \binom{\Lambda-i}{\Lambda\gamma} + \sum_{i=\Lambda\gamma}^{\Lambda-1} b_{\Lambda-i} \binom{i}{\Lambda\gamma} - \sum_{j=1}^{|\mathcal{U}_1|} \sum_{i=\Lambda\gamma}^{a_j-1} \binom{i}{\Lambda\gamma} \\
&\overset{(a)}{=} \sum_{i=1}^{\Lambda-\Lambda\gamma} |\mathcal{U}_i| \binom{\Lambda-i}{\Lambda\gamma} + \sum_{i=\Lambda\gamma}^{\Lambda-1} \sum_{j:a_j \geq i+1}^{|\mathcal{U}_1|} \binom{i}{\Lambda\gamma} \\
&\quad - \sum_{j:a_j-1 \geq \Lambda\gamma}^{|\mathcal{U}_1|} \sum_{i=\Lambda\gamma}^{a_j-1} \binom{i}{\Lambda\gamma} \\
&\overset{(b)}{=} \sum_{i=1}^{\Lambda-\Lambda\gamma} |\mathcal{U}_i| \binom{\Lambda-i}{\Lambda\gamma} + \sum_{j:a_j \geq \Lambda\gamma+1}^{|\mathcal{U}_1|} \sum_{i=\Lambda\gamma}^{a_j-1} \binom{i}{\Lambda\gamma} \\
&\quad - \sum_{j:a_j-1 \geq \Lambda\gamma}^{|\mathcal{U}_1|} \sum_{i=\Lambda\gamma}^{a_j-1} \binom{i}{\Lambda\gamma} \\
&= \sum_{i=1}^{\Lambda-\Lambda\gamma} |\mathcal{U}_i| \binom{\Lambda-i}{\Lambda\gamma}
\end{aligned} \tag{A.22}$$

where step (a) uses the equality $b_{\Lambda-i} = \sum_{j:a_j \geq i+1}^{|\mathcal{U}_1|} 1$, and where step (b) is true as $\sum_{i=a}^{b} \sum_{j:a_j>i}^{c}$ is equivalent to $\sum_{j:a_j>a}^{c} \sum_{i=a}^{a_j}$. Substituting (A.22) into the numerator of (3.18) yields the overall delivery time given in (3.19).

The same performance holds for any $\mathcal{U}$ with the same cache occupancy vector $\mathbf{L}$.

□

A.3 Transition to the Multiple File Request Problem

We here briefly describe how the converse and the scheme presented in the shared cache problem, can fit the multiple file request problem.

Converse In Remark 2 we described the equivalence between the two problems. Based on this equivalence, we will describe how the proof of the converse in Section 3.4 holds in the multiple file request problem with $N_0 = 1$, where now simply some terms carry a different meaning. Firstly, each entry $\mathbf{d}_\lambda$ of the vector defined in equation (3.41) now denotes the vector of file indices requested by user λ. Then we see that Lemma 2 (proved in Section A.2.2) directly applies to the equivalent index coding problem of the multiple file requests problem, where now, for a given permutation σ (see Section A.2.2), all the subfiles placed in row λ — i.e., subfiles $W_{\mathcal{T}_\lambda}^{(\mathbf{d}_{\sigma(\lambda)}(j))}, \forall j \in \mathcal{U}_{\sigma(\lambda)}$ for all $\mathcal{T}_\lambda \subseteq [\Lambda] \setminus \{\sigma(1), \ldots, \sigma(\lambda)\}$ — are obtained from different files requested by the same user, and therefore any two of these subfiles/nodes are not connected by any edge in the side information graph. After these two considerations, the rest of the proof of Lemma 2 is exactly the same. The remaining of the converse consists only of mathematical manipulations which remain unchanged and which yield the same lower bound expression.

Scheme The cache placement phase is identical to the one described in Section 3.3.1, where now each cache λ is associated to the single user λ. In the delivery phase, the scheme now follows directly the steps in Section 3.3.1 applied to the shared-link (single antenna) setting, where now $\boldsymbol{s_\lambda} = \mathcal{U}_\lambda$ (cf. (3.10)). As in the case with shared caches, the scheme consists of L_1 rounds, each serving users

$$\mathcal{R}_j = \bigcup_{\lambda \in [\Lambda]} \big(\mathcal{U}_\lambda(j) : L_\lambda \geq j\big) \tag{A.23}$$

where $\mathcal{U}_\lambda(j)$ is the j-th user in set $\mathcal{U}_\lambda$. The expression in (A.23) now means that the multiple files requested by each user are transmitted in a *time-sharing* manner, and at each round the transmitter serves at most one file per user. Next, equation (3.12) is replaced by

$$\chi_{\mathcal{Q}} = \bigcup_{\lambda \in \mathcal{Q}} \big(\mathcal{U}_\lambda(j) : L_\lambda \geq j\big) \tag{A.24}$$

and then each transmitted vector described in equation (3.13), is substituted by the scalar

$$x_{\chi_{\mathcal{Q}}} = \bigoplus_{\lambda \in \mathcal{Q}: L_\lambda \geq j} W_{\mathcal{Q}\setminus\{\lambda\},1}^{(d_{\mathcal{U}_\lambda(j)})}. \tag{A.25}$$

Finally decoding remains the same, and the calculation of delay follows directly.

Appendix B

Appendix of Chapter 4

B.1 Equalities with Elementary Symmetric Functions

Property 1. *For $e_k(\mathcal{X})$ being the k-th elementary symmetric function in the set $\mathcal{X}$, $|\mathcal{X}| = n$, it holds that*

$$e_k(\mathcal{X}) = e_k(\mathcal{X} \setminus \{x_i\}) + x_i e_{k-1}(\mathcal{X} \setminus \{x_i\}), \tag{B.1}$$

for any $i = 1, 2, \ldots, n$.

Property 1 simply states that the sum of all products of k distinct elements in $\mathcal{X}$ is equal to the sum of all such products not including x_i plus the products including x_i, where the latter can be written as $x_i e_{k-1}(\mathcal{X} \setminus \{x_i\})$.

Property 2. *For $e_k(\mathcal{X})$ being the k-th elementary symmetric function in the set $\mathcal{X} \triangleq \{x_1, \ldots, x_n\}$, $|\mathcal{X}| = n$, and ϕ a subset of $\mathcal{X}$, it holds that*

$$\sum_{x_i \in \{\mathcal{X} \setminus \phi\}} x_i \cdot e_{k-1}(\mathcal{X} \setminus \{x_i\}) = \sum_{q \in C_k^{\mathcal{X}}} \prod_{j=1}^{k} x_{q(j)} \cdot |\{q \setminus \phi\}|, \tag{B.2}$$

for any $k = 1, 2, \ldots, n$.

Proof. Let us introduce the notation $\dot{\boldsymbol{x}}_\theta = \prod_{j=1}^{k} x_{\theta(j)}$. By Property 1, we have that

$$\sum_{x_i \in \{\mathcal{X} \setminus \phi\}} x_i \cdot e_{k-1}(\mathcal{X} \setminus \{x_i\}) = |\mathcal{X} \setminus \phi| \cdot e_k(\mathcal{X}) - \sum_{x_i \in \{\mathcal{X} \setminus \phi\}} e_k(\mathcal{X} \setminus \{x_i\}). \tag{B.3}$$

We can write that $e_k(\mathcal{X} \setminus \{x_i\}) = \sum_{q \subseteq \mathcal{X},\ |q|=k,\ x_i \notin q} \dot{\boldsymbol{x}}_q$. Hence, a particular set $q \subseteq \mathcal{X}$, $|q| = k$, will appear in $e_k(\mathcal{X} \setminus \{x_i\})$ if and only if $x_i \notin q$, which implies that each set q will appear for $|\mathcal{X} \setminus \{\phi, q\}|$ different $x_i \in \mathcal{X}$ in $\sum_{x_i \in \{\mathcal{X} \setminus \phi\}} e_k(\mathcal{X} \setminus \{x_i\})$. Consequently, it follows that

$$\sum_{x_i \in \{\mathcal{X} \setminus \phi\}} e_k(\mathcal{X} \setminus \{x_i\}) = \sum_{\substack{q \subseteq \mathcal{X} \\ |q|=k}} \dot{\boldsymbol{x}}_q \cdot |\mathcal{X} \setminus \{\phi, q\}|. \tag{B.4}$$

Applying (B.4) into (B.3) yields

$$\sum_{x_i \in \{\mathcal{X} \setminus \phi\}} x_i \cdot e_{k-1}(\mathcal{X} \setminus \{x_i\}) = \sum_{q \in C_k^{\mathcal{X}}} \prod_{j=1}^{k} x_{q(j)} \cdot (|\mathcal{X} \setminus \phi| - |\mathcal{X} \setminus \{\phi, q\}|) \tag{B.5}$$

$$\stackrel{(a)}{=} \sum_{q \in C_k^{\mathcal{X}}} \prod_{j=1}^{k} x_{q(j)} \cdot |q \setminus \phi|, \tag{B.6}$$

where in (a) we have applied that

$$|\mathcal{X} \setminus \phi| - |\mathcal{X} \setminus \{\phi, q\}| = |\mathcal{X}| - |\phi| - (|\mathcal{X}| - |\phi \cup q|) = |q \setminus \phi|,$$

which concludes the proof of Property 2. □

Corollary 3. *For $e_k(\mathcal{X})$ being the k-th elementary symmetric function in the set $\mathcal{X} \triangleq \{x_1, \ldots, x_n\}$, $|\mathcal{X}| = n$, it holds that*

$$\sum_{i \in [|\mathcal{X}|]} x_i e_{k-1}(\mathcal{X} \setminus \{x_i\}) = k \cdot e_k(\mathcal{X}), \tag{B.7}$$

for any $k = 1, 2, \ldots, n$.

Proof. Corollary 3 follows directly from Property 2 after setting $\phi = \emptyset$. □

B.2 Convexity of Achievability

In the following, we prove Lemma 4, i.e., that the sequence $\{c_t^{(t)}\}_{t \in \{0,\ldots,\Lambda\}}$ is a decreasing and convex sequence. Since the t-th elementary symmetric polynomial in $\mathbf{L}$ is given by $e_t(\mathbf{L}) \triangleq \sum_{q \in C_{t+1}^{[\Lambda]}} \prod_{j=1}^{t+1} L_{q(j)}$, and $c_t^{(t)} \triangleq \frac{\sum_{q \in C_{t+1}^{[\Lambda]}} \prod_{j=1}^{t+1} L_{q(j)}}{\sum_{q \in C_t^{[\Lambda]}} \prod_{j=1}^{t} L_{q(j)}}$, we can write that

$$c_t^{(t)} = e_1(\mathbf{L}) \qquad \text{if } t = 0 \tag{B.8}$$

$$c_t^{(t)} = \frac{e_{t+1}(\mathbf{L})}{e_t(\mathbf{L})} \qquad \text{if } 1 \leq t \leq \Lambda - 1 \tag{B.9}$$

$$c_t^{(t)} = 0 \qquad \text{if } t = \Lambda \tag{B.10}$$

The proof of Lemma 4 builds on the relation of the coefficients $c_t^{(t)}$ with the elementary symmetric polynomials and the following lemma.

Lemma 13. *Let $\{a_k\}$, $k = [n]$, be a strictly log-concave sequence satisfying that $a_k > 0$ for any $k < n$. Then, the sequence $\{b_k \triangleq \frac{a_{k+1}}{a_k}\}$, $k = [n-1]$, is a decreasing (strictly) convex sequence.*

Proof. Since $\{a_k\}$, $k = [n]$, is a strictly log-concave sequence, it holds that $a_k^2 > a_{k+1}a_{k-1}$. Moreover, since $a_k > 0$ for any $k < n$ by definition of the sequence $\{a_k\}$, we have that

$$a_k^2 > a_{k+1}a_{k-1} \quad \iff \quad \frac{a_k}{a_{k-1}} > \frac{a_{k+1}}{a_k}. \tag{B.11}$$

The right-hand side of (B.11) is equivalent to $b_{k-1} > b_k$, which proves that $\{b_j\}$ is a decreasing sequence. Next, we prove that $\{b_j\}$ is also a convex sequence.

A discrete sequence $\{b_j\}$, $j = [n-1]$, is convex if and only if $2b_j \leq b_{j-1} + b_{j+1}$ for any $j = \{2, \ldots, n-2\}$, which, in our case, is equivalent to prove that

$$2\frac{a_{j+1}}{a_j} \leq \frac{a_j}{a_{j-1}} + \frac{a_{j+2}}{a_{j+1}}, \qquad \forall j = \{2, \ldots, n-2\}. \tag{B.12}$$

Let us now multiply (B.12) by the denominators to obtain

$$2a_{j-1}a_{j+1}^2 \leq a_j^2 a_{j+1} + a_{j-1}a_j a_{j+2} \tag{B.13}$$

$$< a_j^2 a_{j+1} + a_{j-1}a_{j+1}^2, \tag{B.14}$$

which follows from the strict log-concavity of $\{a_k\}$ (i.e., $a_j a_{j+2} < a_{j+1}^2$). Re-ordering terms, we obtain

$$a_{j-1}a_{j+1} < a_j^2, \tag{B.15}$$

which is always true because $\{a_k\}$ is strictly log-concave. Consequently, the sequence $\{b_j\}$ is a strictly convex sequence. □

Thus, we only need to show that the sequence $\{1, \{e_t\}_{t\in[\Lambda]}, 0\}$ is strictly log-concave. If it is strictly log-concave, and upon defining $\{a_k\} = \{1, \{e_t\}_{t\in[\Lambda]}, 0\}$ (such that $b_k \triangleq \frac{a_{k+1}}{a_k} = c_t^{(t)}$), applying Lemma 13 yields that $c_t^{(t)}$ is a strictly convex sequence, which will conclude the proof of Lemma 4.

In order to prove this, let us first introduce the *elementary symmetric means* E_k, which are defined as

$$E_k(\mathbf{L}) \triangleq \frac{e_k(\mathbf{L})}{\binom{n}{k}}, \tag{B.16}$$

where $\binom{n}{k}$ is the number of summands in $e_k(\mathbf{L})$ for a set of $|\mathbf{L}| = n$ elements (in our case, $n = \Lambda$). Hereinafter, we omit the dependence of e_k and E_k on $\mathbf{L}$ because it is the same set for any e_k, E_k considered in the following.

It was shown by Newton [97] that, for any n-tuple of non-negative numbers, it holds that the sequence $\{E_k\}_{k\in[n]}$ is a log-concave sequence, and thus

$$E_k^2 \geq E_{k-1}E_{k+1}, \quad k \in \{2, \ldots, n-1\}, \tag{B.17}$$

where the inequality is strict unless all the elements of the n-tuple coincide.

In order to prove log-concavity, we first obtain from (B.16)-(B.17) that

$$E_t^2 \geq E_{t-1}E_{t+1} \quad \Longleftrightarrow \quad \frac{e_t^2}{\binom{n}{t}^2} \geq \frac{e_{t-1}e_{t+1}}{\binom{n}{t-1}\binom{n}{t+1}}, \tag{B.18}$$

for any $1 < t < n$. Then, we can write that

$$e_t^2 \geq \frac{\binom{n}{t}^2}{\binom{n}{t-1}\binom{n}{t+1}} e_{t-1}e_{t+1} \tag{B.19}$$

$$> e_{t-1}e_{t+1}, \tag{B.20}$$

which implies that $\{e_t\}_{1\leq t\leq \Lambda}$ is strictly log-concave, and where the last step follows from the strict log-concavity of the binomial coefficient [98]. It remains to prove that $e_1^2 > 1 \times e_2 = e_2$ and that $e_\Lambda^2 > e_{\Lambda-1} \times 0 = 0$. The latest is always true since $e_\Lambda = \prod_{j=1}^{\Lambda} L_j > 0$ because $L_j \geq 1$. For the former ($e_1^2 > e_2$), let us recall the Maclaurin inequalities [99], which state that

$$E_1 \geq \sqrt{E_2} \geq \sqrt[3]{E_3} \geq \cdots \geq \sqrt[n]{E_n}, \tag{B.21}$$

with equality only if all the L_j for any $j \in [n]$ coincide. Let us focus on the first inequality. Since $n = \Lambda$ and $L_j \geq 1$ for any j, it follows that

$$E_1 \geq \sqrt{E_2} \quad \Longleftrightarrow \quad \frac{e_1^2}{\binom{\Lambda}{1}^2} \geq \frac{e_2}{\binom{\Lambda}{2}} \quad \Longleftrightarrow \quad \frac{e_1^2}{\Lambda^2} \geq \frac{e_2}{\frac{\Lambda(\Lambda-1)}{2}} \quad \Longleftrightarrow \quad e_1^2 \geq \frac{2\Lambda}{\Lambda-1}e_2. \tag{B.22}$$

Since $e_1^2 \geq \frac{2\Lambda}{\Lambda-1}e_2 > e_2$, we obtain that the sequence $\{1, \{e_t\}_{1\leq t\leq \Lambda}, 0\}$ is strictly log-concave. Thus, applying Lemma 13 yields that $c_t^{(t)}$ is a strictly convex sequence, which concludes the proof of Lemma 4.

B.3 Proof of Lemma 6

From (4.35) and (4.32) we can write $T_{lb,p}(\boldsymbol{Z}, \mathbf{L})$ as

$$T_{lb,p}(\boldsymbol{Z}, \mathbf{L}) = \underbrace{\frac{1}{|\mathcal{D}_{wc}| \sum_{\sigma\in\mathcal{S}_\Lambda} \prod_{j=1}^{p} L_{\sigma(\Lambda-p+j)}}}_{\triangleq \mathcal{D}_p} \underbrace{\sum_{\mathbf{d}\in\mathcal{D}_{wc}} \sum_{\sigma\in\mathcal{S}_\Lambda} \mathcal{Q}_\sigma^{(p)} T_{lb,\sigma}(\boldsymbol{Z}, \mathbf{d}, \mathbf{L})}_{\triangleq \mathcal{N}_p}. \tag{B.23}$$

where $\mathcal{Q}_\sigma^{(p)} \triangleq \prod_{j=1}^{p} L_{\sigma(\Lambda-p+j)}$ and

$$T_{lb,\sigma}(\boldsymbol{Z}, \mathbf{d}, \mathbf{L}) \triangleq \sum_{\lambda=1}^{\Lambda} \sum_{\ell=1}^{L_{\sigma(\lambda)}} \sum_{\tau_\lambda \subseteq [\Lambda]\setminus\{\sigma(1),\ldots,\sigma(\lambda)\}} |W_{\tau_\lambda}^{(\mathbf{d}_{\sigma(\lambda)}(\ell))}|. \tag{B.24}$$

First, we rewrite $\mathcal{D}_p$ (defined in (B.23)) in a more suitable form as

$$\begin{aligned}\mathcal{D}_p &= P(N,K)\sum_{\sigma\in\mathcal{S}_\Lambda}\prod_{j=1}^{p}L_{\sigma(\Lambda-p+j)}\\ &= P(N,K)\sum_{v\in\mathcal{S}_{\Lambda-p}}\sum_{q\in\mathcal{S}_p}\prod_{j=1}^{p}L_{q(j)}\\ &= P(N,K)(\Lambda-p)!p!\sum_{q\in C_p^{[\Lambda]}}\prod_{j=1}^{p}L_{q(j)},\end{aligned} \tag{B.25}$$

which follows from basic mathematical manipulations.

For any $n\in[N]$ and any $\tau\subseteq[\Lambda]$, our goal is now to evaluate the coefficient that multiplies each $|W_\tau^{(n)}|$ in $\mathcal{N}_p$ from (B.23), which we denote here by $g_{n,\tau}^{(p)}$. We first state the following trivial fact.

Fact 2. For any $n\in[N]$ and any $\tau\subseteq[\Lambda]$, if $|W_\tau^{(n)}|$ appears in $T_{lb,\sigma}(\boldsymbol{Z},\mathbf{d},\mathbf{L})$ for some $\mathbf{d}\in\mathcal{D}_{wc}$ and some $\sigma\in S_\Lambda$, then it only appears once for all $\lambda\in[\Lambda],\ \ell\in[L_\lambda]$.

Let us focus on a demand vector $\mathbf{d}'$ such that subfile $|W_\tau^{(n)}|$ is requested by a certain user k' associated to cache λ', i.e. $k'\in\mathcal{U}_{\lambda'}$ and $d_{k'}=n$. Our objective is now to evaluate the coefficient of $|W_\tau^{(d_{k'})}|$ in

$$T_{lb}(\boldsymbol{Z},\mathbf{d}',\mathbf{L})\triangleq\sum_{\sigma\in\mathcal{S}_\Lambda}\mathcal{Q}_\sigma^{(p)}T_{lb,\sigma}(\boldsymbol{Z},\mathbf{d}',\mathbf{L}). \tag{B.26}$$

In this respect, we need to identify those permutations $\sigma\in S_\Lambda$ for which $|W_\tau^{(d_{k'})}|$ appears in $T_{lb,\sigma}(\boldsymbol{Z},\mathbf{d}',\mathbf{L})$.
From the expression of $T_{lb,\sigma}(\boldsymbol{Z},\mathbf{d}',\mathbf{L})$ in (B.24), the following proposition holds.

Proposition 6. *The permutations σ for which $|W_\tau^{(d_{k'})}|$ appears in $T_{lb,\sigma}(\boldsymbol{Z},\mathbf{d}',\mathbf{L})$ are such that λ' appears in the permutation σ before any element of the set τ. We will refer to such permutations as valid permutations.*

Example: Consider $W_{1,2}^{(3)}$, where $d_k=3$ for $k\in\mathcal{U}_3$ and $\tau=\{1,2\}$. This subfile will not appear in $T_{lb,\sigma}(\boldsymbol{Z},\mathbf{d}',\mathbf{L})$ for permutations σ $(1,2,3),(2,1,3),(1,3,2)$ and $(2,3,1)$, while it will appear for permutations $(3,1,2)$ and $(3,2,1)$.
Because of (B.26), when we identify any valid permutation σ we will say that we identified $\mathcal{Q}_\sigma^{(p)}$ such permutations.

Next, we need to split the proof in two cases. First, we consider the case when $|\tau|\geq p$, and afterwards we focus on the case $|\tau|<p$. In the following, we make use of the notation $\dot{\boldsymbol{L}}_q\triangleq\prod_{j=1}^{|q|}L_{q(j)}$ for any $q\subseteq[\Lambda]$.

The $|\tau| \geq p$ case

With Proposition 6 at hand, we notice that, if λ' appears in any one of the last p positions of a permutation σ, then there will be for sure an element of τ (recall that $|\tau| \geq p$) which will precede λ'. It follows that, in this case, any valid permutation does not have λ' in the last p positions. Thus, the number of ways in which we can fill these last p positions, recalling that permutation σ accounts for $\mathcal{Q}_\sigma^{(p)}$ times, is

$$p! \sum_{q \in C_p^{[\Lambda]\setminus\{\lambda'\}}} \dot{\boldsymbol{L}}_q. \tag{B.27}$$

We continue with the following proposition.

Proposition 7. *For each of the ways in which we can fill the last p positions of permutation σ, the number of ways in which we can fill the first $\Lambda - p$ positions so that σ is a valid permutation as stated in Proposition 6 is*

$$\frac{\binom{\Lambda-p}{|\tau\setminus q|+1}}{\binom{\Lambda-p-1}{|\tau\setminus q|}}(\Lambda - p - 1)! \tag{B.28}$$

Proof. Let us denote the set of elements in the last p positions of σ by q, and let us consider the case when λ' is in position r for some $r \in [\Lambda - p]$. In this case, we have that in the first $r-1$ positions we cannot place any of the elements in q, we cannot place λ', and we cannot put any element of τ that is not in q. It then follows that, for any q, there are

$$P(\Lambda - p - 1 - |\tau \setminus q|)(\Lambda - p - r)!$$

ways in which we can fill the first $\Lambda - p$ positions of σ with λ' in position r. Considering all possible r values we have

$$\sum_{r=1}^{\Lambda-p} P(\Lambda - p - 1 - |\tau \setminus q|, r-1)(\Lambda - p - r)!, \tag{B.29}$$

which can be manipulated such that

$$\begin{aligned}
&\sum_{r=1}^{\Lambda-p} P(\Lambda - p - 1 - |\tau \setminus q|, r-1)(\Lambda - p - r)! \\
=&\sum_{r=1}^{\Lambda-p} \frac{(\Lambda - p - 1 - |\tau \setminus q|)!(\Lambda - p - r)!|\tau \setminus q|!(\Lambda - p - 1)!}{(\Lambda - p - |\tau \setminus q| - r)!|\tau \setminus q|!(\Lambda - p - 1)!} \\
=&\sum_{r=1}^{\Lambda-p} \frac{\binom{\Lambda-p-r}{|\tau\setminus q|}}{\binom{\Lambda-p-1}{|\tau\setminus q|}}(\Lambda - p - 1)! \\
=&\frac{\binom{\Lambda-p}{|\tau\setminus q|+1}}{\binom{\Lambda-p-1}{|\tau\setminus q|}}(\Lambda - p - 1)!,
\end{aligned} \tag{B.30}$$

which matches (B.28). □

Combining equation (B.27) and Proposition 7, the total number of valid permutations σ (with repetition given by $\mathcal{Q}_\sigma^{(p)}$) is

$$p! \sum_{q \in C_p^{[\Lambda]\setminus\{\lambda'\}}} \dot{\boldsymbol{L}}_q \frac{\binom{\Lambda-p}{|\tau\setminus q|+1}}{\binom{\Lambda-p-1}{|\tau\setminus q|}} (\Lambda-p-1)! \tag{B.31}$$

We now proceed to evaluate the total number of demands for which subfile $W_\tau^{(d_{k'})}$ is requested.

It is easy to see that the total number of demands with $d_{k'} = n$ is $P(N-1, K-1)$. If user k' is associated to any of the caches in set τ, then subfile $W_\tau^{(d'_k)}$ will not be requested to the server, since it is already stored in cache λ'. Thus, $W_\tau^{(n)}$ will be requested to the server only if $\lambda' \in [\Lambda] \setminus \{\tau\}$. It follows that the total number of times that subfile $W_\tau^{(n)}$ is requested among all possible demand vectors is

$$P(N-1, K-1) \sum_{\lambda \in [\Lambda]\setminus\{\tau\}} L_\lambda. \tag{B.32}$$

From equations (B.31) and (B.32), we have that the coefficient of $|W_\tau^{(n)}|$ in $\mathcal{N}_p$ can be written as

$$g_{n,\tau}^{(p)} = P(N-1,K-1) \cdot \sum_{\lambda \in [\Lambda]\setminus\{\tau\}} L_\lambda \cdot p! \cdot \sum_{q \in C_p^{[\Lambda]\setminus\{\lambda\}}} \left(\dot{\boldsymbol{L}}_q \times \frac{\binom{\Lambda-p}{|\tau\setminus q|+1}}{\binom{\Lambda-p-1}{|\tau\setminus q|}} (\Lambda-p-1)! \right). \tag{B.33}$$

Finally, we obtain the coefficient of any $|W_\tau^{(n)}|$ with $|\tau| \geq p$ in $T_{lb,p}(\boldsymbol{Z}, \mathbf{L})$, i.e., $\frac{g_{n,\tau}^{(p)}}{\mathcal{D}_p}$ (cf. (B.23)), which can be rewritten as

$$\begin{aligned}
\frac{g_{n,\tau}^{(p)}}{\mathcal{D}_p} =& \frac{P(N-1,K-1) \cdot \sum_{\lambda \in [\Lambda]\setminus\{\tau\}} L_\lambda \cdot p! \cdot \sum_{q \in C_p^{[\Lambda]\setminus\{\lambda\}}} \left(\dot{\boldsymbol{L}}_q \times \frac{\binom{\Lambda-p}{|\tau\setminus q|+1}}{\binom{\Lambda-p-1}{|\tau\setminus q|}} (\Lambda-p-1)! \right)}{P(N,K)(\Lambda-p)!p! \sum_{\ell \in C_p^{[\Lambda]}} \dot{\boldsymbol{L}}_\ell} \\
&\overset{(a)}{=} \frac{1}{N \cdot \sum_{\ell \in C_p^{[\Lambda]}} \dot{\boldsymbol{L}}_\ell} \sum_{\lambda \in [\Lambda]\setminus\{\tau\}} L_\lambda \sum_{q \in C_p^{[\Lambda]\setminus\{\lambda\}}} \dot{\boldsymbol{L}}_q \cdot \frac{1}{|\tau \setminus q| + 1} \\
&\overset{(b)}{=} \frac{\sum_{q \in C_{p+1}^{[\Lambda]}} \dot{\boldsymbol{L}}_q \cdot \frac{|q\setminus\tau|}{|\tau\setminus q|+1}}{N \cdot \sum_{\ell \in C_p^{[\Lambda]}} \dot{\boldsymbol{L}}_\ell} \\
&\overset{(c)}{=} \frac{\sum_{q \in C_{p+1}^{[\Lambda]}} \dot{\boldsymbol{L}}_q \cdot \frac{p+1-|q\cap\tau|}{|\tau|+1-|q\cap\tau|}}{N \cdot \sum_{\ell \in C_p^{[\Lambda]}} \dot{\boldsymbol{L}}_\ell},
\end{aligned} \tag{B.34}$$

where (a) follows from basic mathematical manipulations, (b) follows from Property 2 and the fact that for any $\lambda \in [\Lambda] \setminus \{\tau\}$ and $q \in C_p^{[\Lambda]\setminus\{\lambda\}}$ we have $|\tau \setminus q| = |\tau \setminus \{q \cup \{\lambda\}\}|$, and (c) follows from the fact that $|q \setminus \tau| = |q| - |q \cap \tau|$ as well as $|\tau \setminus q| = |\tau| - |q \cap \tau|$. Defining $c_{n,\tau}^{(p)} = N \frac{g_{n,\tau}^{(p)}}{\mathcal{D}_p}$ proves the lemma for the case $\tau \geq p$.

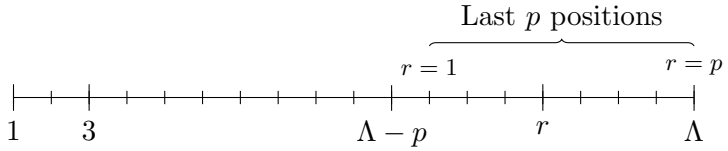


Figure B.1 – Illustration of the meaning of index r.

The $|\tau| < p$ case

As for the $|\tau| \geq p$ case, we focus on a demand vector such that $W_\tau^{(n)}$ is requested by a certain user.

The number of permutations such that λ is not in the last p positions is the same as for the case $|\tau| = j \geq p$, i.e., (B.34). Let us denote such value as $c_{\tau,start}^{(p)} = \frac{g_{n,\tau}^{(p)}}{\mathcal{D}_p}$. But now, since $j < p$, λ can be also located up to the $\Lambda - j$ position, i.e., it can appear in some of the last p positions. Then, the coefficient can be written as

$$c_\tau^{(p)} \triangleq c_{\tau,start}^{(p)} + c_{\tau,end}^{(p)}, \tag{B.35}$$

where $c_{\tau,end}^{(p)}$ is given by those permutations where λ is in one of the last p positions. In order to obtain $c_{\tau,end}^{(p)}$, let us first fix the position of λ, such that r denotes in which of the last p positions is located λ. Hence, $r = 1$ implies that λ is in the last $\Lambda - p + 1$ position, whereas $r = p$ implies that λ is in the last position (see Fig. B.1 for a visual explanation).

If λ is in the last p positions, Proposition 6 implies that all the values in τ must also be in those last p positions, and in particular in the positions $\{\Lambda - p + r + 1, \ldots, \Lambda\}$. Now, the remaining $p - j - 1$ positions can be filled with the other $\Lambda - j - 1$ caches. Let us consider a particular set q of p caches filling the last p positions. We can write such set as

$$q \triangleq \{\lambda \cup \tau \cup s\}, \tag{B.36}$$

where $\lambda \cap \tau = \varnothing$, and $s \in C_{p-j-1}^{[\Lambda]\backslash\{\lambda\cup\tau\}}$. Note that $|[\Lambda]\backslash\{\lambda \cup \tau\}| = \Lambda - j - 1$. Then, the numerator of the coefficient $c_{\tau,end}^{(p)}$ is given by

$$\begin{aligned} c_{\tau,end}^{(p),num} = {} & \underbrace{P(N-1,K-1) \sum_{\lambda\in[\Lambda]\backslash\{\tau\}} L_\lambda}_{\text{Times that } W_\tau^{(n)} \text{ is requested (B.32)}} \underbrace{\sum_{s\in C_{p-j-1}^{[\Lambda]\backslash\{\lambda\cup\tau\}}}}_{\substack{\text{All possible} \\ \text{combinations} \\ \text{of caches} \\ \text{in last } p \text{ positions} \\ \text{apart from } \lambda,\tau}} \overbrace{\underbrace{L_\lambda \dot{\boldsymbol{L}}_\tau \dot{\boldsymbol{L}}_s}_{=\dot{\boldsymbol{L}}_q}}^{} \times \\ & \times \underbrace{\sum_{r=1}^{p-j}}_{\substack{\text{Possible} \\ \text{positions} \\ \text{of } \lambda}} \underbrace{P(p-r,j)}_{\substack{\text{Ways of putting} \\ \tau \text{ in last} \\ p-r \text{ positions}}} \underbrace{(p-j-1)!}_{\substack{\text{For every } \tau,\lambda, \\ \text{ways of filling} \\ \text{the other} \\ p-j-1 \text{ positions}}} \underbrace{(\Lambda-p)!}_{\substack{\text{Filling the} \\ \text{first } \Lambda-p \\ \text{positions}}} \end{aligned} \tag{B.37}$$

Adding the denominator (as before), we can simplify as

$$c_{\tau,end}^{(p)} = \frac{c_{\tau,end}^{(p),num}}{P(N,K)p!(\Lambda-p)!\sum_{\ell\in C_p^{[\Lambda]}}\dot{\boldsymbol{L}}_\ell} \tag{B.38}$$

$$\overset{(a)}{=} \frac{\sum_{\lambda\in[\Lambda]\setminus\{\tau\}} L_\lambda \sum_{s\in C_{p-j-1}^{[\Lambda]\setminus\{\lambda\cup\tau\}}} L_\lambda \dot{\boldsymbol{L}}_\tau \dot{\boldsymbol{L}}_s \sum_{r=1}^{p-j} P(p-r,j)(p-j-1)!}{Np!\sum_{\ell\in C_p^{[\Lambda]}}\dot{\boldsymbol{L}}_\ell} \tag{B.39}$$

$$\overset{(b)}{=} \frac{\sum_{\lambda\in[\Lambda]\setminus\{\tau\}} L_\lambda \sum_{s\in C_{p-j-1}^{[\Lambda]\setminus\{\lambda\cup\tau\}}} L_\lambda \dot{\boldsymbol{L}}_\tau \dot{\boldsymbol{L}}_s \frac{p!}{j+1}}{Np!\sum_{\ell\in C_p^{[\Lambda]}}\dot{\boldsymbol{L}}_\ell}, \tag{B.40}$$

where in the (a) we have applied that $\frac{P(N-1,K-1)}{P(N,K)} = \frac{1}{N}$ and canceled out $(\Lambda-p)!$, and in (b) we have applied that

$$\sum_{r=1}^{p-j} P(p-r,j)(p-j-1)! = (p-j-1)!\sum_{r=1}^{p-j}\frac{(p-r)!}{(p-r-j)!} \tag{B.41}$$

$$= \frac{p!}{j+1}. \tag{B.42}$$

Then, by recalling that $j = |\tau|$, it follows that

$$c_{\tau,end}^{(p)} = \frac{1}{|\tau|+1}\frac{\sum_{\lambda\in[\Lambda]\setminus\{\tau\}} L_\lambda \sum_{s\in C_{p-j-1}^{[\Lambda]\setminus\{\lambda\cup\tau\}}} L_\lambda \dot{\boldsymbol{L}}_\tau \dot{\boldsymbol{L}}_s}{N\sum_{\ell\in C_p^{[\Lambda]}}\dot{\boldsymbol{L}}_\ell} \tag{B.43}$$

$$= \frac{1}{N\sum_{\ell\in C_p^{[\Lambda]}}\dot{\boldsymbol{L}}_\ell}\cdot\frac{1}{|\tau|+1}\dot{\boldsymbol{L}}_\tau \sum_{s\in C_{p-j}^{[\Lambda]\setminus\{\tau\}}}\Big(\dot{\boldsymbol{L}}_s\cdot\sum_{i=1}^{p-j} L_{s(i)}\Big). \tag{B.44}$$

Thus, from (B.34) and (B.44), the total coefficient $c_\tau^{(p)} \triangleq c_{\tau,start}^{(p)} + c_{\tau,end}^{(p)}$ is then given by

$$c_\tau^{(p)} = \frac{1}{N\sum_{\ell\in C_p^{[\Lambda]}}\dot{\boldsymbol{L}}_\ell}\Bigg(\sum_{q\in C_{p+1}^{[\Lambda]}}\dot{\boldsymbol{L}}_q\frac{p+1-|q\cap\tau|}{|\tau|+1-|q\cap\tau|} + \frac{1}{|\tau|+1}\dot{\boldsymbol{L}}_\tau\sum_{s\in C_{p-j}^{[\Lambda]\setminus\{\tau\}}}\Big(\dot{\boldsymbol{L}}_s\cdot\sum_{i=1}^{p-j}L_{s(i)}\Big)\Bigg), \tag{B.45}$$

which concludes the proof of Lemma 6. Note that, for the case where $j \geq p$, it holds that $c_{\tau,end}^{(p)} = 0$, and hence $c_\tau^{(p)} = c_{\tau,start}^{(p)}$, which allows us to consider (B.45) for any value of $|\tau|$. □

B.4 Proof of Proposition 2

In the following, we prove that the sequence $\left\{\tilde{c}^{(p)}_{\tau^\star_j}\right\}$ is a decreasing sequence in $j \in [\Lambda]_0 = \{0 \cup [\Lambda]\}$, where we recall that $\tilde{c}^{(p)}_\tau$ is given by

$$\tilde{c}^{(p)}_\tau \triangleq \frac{\sum_{q \in C^{[\Lambda]}_{p+1}} \frac{p+1-|q\cap\tau|}{|\tau|+1-|q\cap\tau|} \prod_{j=1}^{p+1} L_{q(j)}}{\sum_{\ell \in C^{[\Lambda]}_{p}} \prod_{j=1}^{p} L_{\ell(j)}},$$

and $\tau^\star_j$ denotes $\tau^\star_j \triangleq \operatorname{argmin}_{\tau \subset [\Lambda]_0, |\tau|=j} \tilde{c}^{(p)}_\tau$. Since the denominator of $\tilde{c}^{(p)}_\tau$ is the same for any τ, we focus on the numerator. First, let us denote the numerator of $\tilde{c}^{(p)}_\tau$ by $A(p,\tau)$, such that

$$A(p,\tau) \triangleq \sum_{q \in C^{[\Lambda]}_{p+1}} \frac{p+1-|q\cap\tau|}{|\tau|+1-|q\cap\tau|} \prod_{j=1}^{p+1} L_{q(j)}. \tag{B.46}$$

Hence, we need to prove that, for any $0 \le j \le \Lambda - 1$, it holds that

$$A(p,\ \tau^\star_j) > A(p,\ \tau^\star_{j+1}). \tag{B.47}$$

Let us consider an arbitrary j, $0 \le j \le \Lambda - 1$. We select a set τ' with cardinality $j+1$ which comprises $\tau^\star_j$, i.e., we can write τ' as $\tau' = \{\tau^\star_j \cup r\}$, where $r \in \{[\Lambda] \setminus \tau^\star_j\}$. Then, it follows from (B.46) that

$$A(p,\tau') = \sum_{q \in C^{[\Lambda]}_{p+1}} \frac{p+1-|q\cap\{\tau^\star_j \cup r\}|}{(j+1)+1-|q\cap\{\tau^\star_j \cup r\}|} \prod_{j=1}^{p+1} L_{q(j)}. \tag{B.48}$$

Note that $\prod_{j=1}^{p+1} L_{q(j)}$ is independent of $\tau', \tau^\star_j$. Furthermore, it holds that

$$\frac{p+1-|q\cap\{\tau^\star_j \cup r\}|}{(j+1)+1-|q\cap\{\tau^\star_j \cup r\}|} < \frac{p+1-|q\cap\tau^\star_j|}{j+1-|q\cap\tau^\star_j|} \tag{B.49}$$

for any $p \in \Lambda$, $r \in \{[\Lambda] \setminus \tau^\star_j\}$, $q \in C^{[\Lambda]}_{p+1}$. Merging (B.48) and (B.49) yields

$$A(p,\tau') < \sum_{q \in C^{[\Lambda]}_{p+1}} \frac{p+1-|q\cap\tau^\star_j|}{|\tau^\star_j|+1-|q\cap\tau^\star_j|} \prod_{j=1}^{p+1} L_{q(j)} = A(p,\tau^\star_j). \tag{B.50}$$

By definition, $A(p,\tau') \ge A(p,\tau^\star_{j+1})$ for any τ' such that $|\tau'| = j+1$. Thus,

$$A(p,\tau^\star_{j+1}) \le A(p,\tau') < A(p,\tau^\star_j), \tag{B.51}$$

which concludes the proof of Proposition 2. □

B.5 Proof of Lemma 7

In this appendix, we obtain which set τ of cardinality $|\tau| = j$ minimizes $\tilde{c}_\tau^{(p)}$. For any set q, we recover the notation $\dot{\boldsymbol{L}}_q \triangleq \prod_{j=1}^{|q|} L_{q(j)}$ for the sake of readability. Let us start by recalling that $\tilde{c}_\tau^{(p)}$ is defined as

$$\tilde{c}_\tau^{(p)} \triangleq \frac{\sum_{q \in C_{p+1}^{[\Lambda]}} \dot{\boldsymbol{L}}_q \frac{p+1-|q\cap\tau|}{|\tau|+1-|q\cap\tau|}}{\sum_{\ell \in C_p^{[\Lambda]}} \dot{\boldsymbol{L}}_\ell}. \tag{B.52}$$

For the case of $j = 0$, the only possible set τ is the empty set, whereas for the case $j = p$ we can see from (B.52) that all $\tilde{c}_\tau^{(p)}$ for any $\tau : |\tau| = p$ have the same value, and thus any $\tau : |\tau| = p$ is a solution of the optimization problem. We select $\tau_p^\star = \{1,\ 2,\ \ldots,\ p\}$ without loss of generality. In the following, we focus on the cases where $j \in \{[\Lambda] \setminus p\}$.

We can re-write (B.52) as follows: Since $\frac{p+1-|q\cap\tau|}{|\tau|+1-|q\cap\tau|} = 1 + \frac{p-|\tau|}{|\tau|+1-|q\cap\tau|}$, we have that

$$\tilde{c}_\tau^{(p)} = \frac{\sum_{q \in C_{p+1}^{[\Lambda]}} \dot{\boldsymbol{L}}_q}{\sum_{\ell \in C_p^{[\Lambda]}} \dot{\boldsymbol{L}}_\ell} + \frac{p-|\tau|}{\sum_{\ell \in C_p^{[\Lambda]}} \dot{\boldsymbol{L}}_\ell} \Big(\underbrace{\sum_{q \in C_{p+1}^{[\Lambda]}} \dot{\boldsymbol{L}}_q \frac{1}{|\tau|+1-|q\cap\tau|}}_{a_\tau^{(p)}} \Big). \tag{B.53}$$

Let us denote the $j = |\tau|$ elements of the set τ as $\{\tau_1, \ldots, \tau_j\}$, $\tau_i \in [\Lambda]\ \forall i \in [j]$. For any τ such that $|\tau| = j$, the only term in (B.53) that differs with respect to any other τ' of the same cardinality is the sum denoted by $a_\tau^{(p)}$. Then, we prove Lemma 7 by means of finding the subset τ that optimizes $a_\tau^{(p)}$. Note that, since the sign of $(p - |\tau|)$ is different whether $p > |\tau|$ or not, we have that

$$\underset{\tau \subseteq [\Lambda],\, |\tau| = j}{\operatorname{argmin}} \tilde{c}_\tau^{(p)} = \underset{\tau \subseteq [\Lambda],\, |\tau| = j}{\operatorname{argmin}} \operatorname{sgn}(p-j) a_\tau^{(p)}. \tag{B.54}$$

Hereinafter, we also use the notation $\tau_{[n]} \triangleq \{\tau_1, \ldots, \tau_n\}$ for any $n \leq j = |\tau|$. We break the problem of (B.54) in different sub-problems, which allows us to simplify the proof. Specifically, we first obtain the element τ_j that minimizes $a_\tau^{(p)}$ for each given set $\tau_{[j-1]} \subset [\Lambda]$. Let us denote this value as $\tau_j^\star(\tau_{[j-1]})$, i.e., $\tau_j^\star(\tau_{[j-1]}) \triangleq \operatorname{argmin}_{\tau_j \in \{[\Lambda] \setminus \tau_{[j-1]}\}} \operatorname{sgn}(p-j) a_\tau^{(p)}$. Note that, since there are $\binom{\Lambda}{j-1}$ different $\tau_{[j-1]}$, we obtain $\binom{\Lambda}{j-1}$ different $\tau_j^\star(\tau_{[j-1]})$.

Next, we consider the following element of τ, i.e., τ_{j-1}. It turns out that

$$\tau_{j-1}^\star(\tau_{[j-2]}) \triangleq \underset{\tau_{j-1}, \tau_j \in \{[\Lambda] \setminus \tau_{[j-2]}\}}{\operatorname{argmin}} \operatorname{sgn}(p-j) a_\tau^{(p)} = \underset{\tau_{j-1} \in \{[\Lambda] \setminus \{\tau_{[j-2]}, \tau_j^\star\}\}}{\operatorname{argmin}} \operatorname{sgn}(p-j) a_\tau^{(p)}. \tag{B.55}$$

Then, defining $\tau_n^\star(\tau_{[n-1]})$ iteratively over all the elements of τ as in (B.55) yields $\tau_1^\star \triangleq \operatorname{argmin}_{\tau_1 \in \{[\Lambda] \setminus \{\tau_2^\star, \ldots, \tau_j^\star\}\}} \operatorname{sgn}(p-j) a_\tau^{(p)}$, and thus

$$\underset{\tau \subseteq [\Lambda],\, |\tau| = j}{\operatorname{argmin}} \tilde{c}_\tau^{(p)} = \{\tau_1^\star, \tau_2^\star, \ldots, \tau_j^\star\}. \tag{B.56}$$

The following lemma is key for the derivation of Lemma 7.

Lemma 14. *Let us consider a given set* $\{\tau_1, \ldots, \tau_{j-1}\} \subset [\Lambda]$. *Then, it holds that*

$$\tau_j^\star(\tau_{[j-1]}) \triangleq \underset{\tau_j \in \{[\Lambda]\backslash\tau_{[j-1]}\}}{\operatorname{argmin}} \operatorname{sgn}(p-j) a_\tau^{(p)} = \underset{\tau_j \in \{[\Lambda]\backslash\tau_{[j-1]}\}}{\operatorname{argmin}} \operatorname{sgn}(p-j) b_\tau^{(p)} \tag{B.57}$$

where $b_\tau^{(p)}$ *is defined as*

$$b_\tau^{(p)} \triangleq \sum_{\ell=0}^{\min(p+1,j)} \frac{-1}{(j-\ell)(j-\ell+1)} \sum_{\substack{\mu \subseteq \{\tau\backslash\tau_j\} \\ |\mu|=\ell}} \dot{\boldsymbol{L}}_\mu \sum_{q \in C_{p+1-\ell}^{[\Lambda]\backslash\{\tau\}}} \dot{\boldsymbol{L}}_q. \tag{B.58}$$

Proof. The proof is relegated to Section B.5.1. □

Let us now split the proof for the cases where $p < j$ and $p > j$.

Case $p > j$

Note that the sum $\sum_{\substack{\mu \subseteq \{\tau\backslash\tau_j\} \\ |\mu|=\ell}} \dot{\boldsymbol{L}}_\mu$ in (B.57) depends only on the $j-1$ elements in τ that are assumed to be fixed in this step. Furthermore, all the terms in (B.58) are negative and $p > j$ implies that $\operatorname{sgn}(p-j) = 1$. Then, in order to minimize (B.58), we need to make $\sum_{q \in C_{p+1-\ell}^{[\Lambda]\backslash\{\tau\}}} \dot{\boldsymbol{L}}_q$ as big as possible. Since τ_j is the term that we *remove* from the sum (note that $q \subset [\Lambda]\backslash\{\tau\}$), this is achieved by selecting the cache in $\left\{[\Lambda]\backslash\tau_{[j-1]}\right\}$ with the smallest L. Given that we have ordered the caches such that $L_1 \geq \cdots \geq L_\Lambda$, we obtain that

$$\tau_j^\star(\tau_{[j-1]}) = \underset{\tau_j \in \{[\Lambda]\backslash\tau_{[j-1]}\}}{\operatorname{argmin}} b_\tau^{(p)} = \underset{\tau_j \in \{[\Lambda]\backslash\tau_{[j-1]}\}}{\operatorname{argmin}} L_{\tau_j} = \underset{\tau_j \in \{[\Lambda]\backslash\tau_{[j-1]}\}}{\max} \tau_j. \tag{B.59}$$

Therefore, if we obtain (B.59) for any possible $\{\tau_1, \ldots, \tau_{j-1}\}$, we obtain that the optimal τ_j is cache Λ (because L_Λ is the smallest L_i). Applying this reasoning iteratively, we obtain that

$$\underset{\tau \subseteq [\Lambda],\, |\tau|=j}{\operatorname{argmin}} \tilde{c}_\tau^{(p)} = \{\Lambda - j + 1, \ldots, \Lambda\}, \tag{B.60}$$

which concludes the proof for $p > j$.

Case $p < j$

In this case, the difference is that $\operatorname{sgn}(p-j) = -1$. With the change of sign, all the terms are positive, and thus we aim to maximize (B.58). Then, we need to make $\sum_{q \in C_{p+1-\ell}^{[\Lambda]\backslash\{\tau\}}} \dot{\boldsymbol{L}}_q$ as small as possible. Since τ_j is the term that we *remove* from the sum, this is achieved by

selecting the cache in $\left\{[\Lambda]\backslash\tau_{[j-1]}\right\}$ with the biggest L. Thus, applying the same reasoning as for the case $p > j$, we obtain that

$$\tau_j^\star(\tau_{[j-1]}) = \underset{\tau_j\in\{[\Lambda]\backslash\tau_{[j-1]}\}}{\operatorname{argmax}} b_\tau^{(p)} = \underset{\tau_j\in\{[\Lambda]\backslash\tau_{[j-1]}\}}{\operatorname{argmax}} L_{\tau_j} = \underset{\tau_j\in\{[\Lambda]\backslash\tau_{[j-1]}\}}{\min} \tau_j. \tag{B.61}$$

Therefore, applying this reasoning iteratively, we obtain that

$$\underset{\tau\subseteq[\Lambda],\,|\tau|=j}{\operatorname{argmin}} \tilde{c}_\tau^{(p)} = \{1, 2, \ldots, j\}, \tag{B.62}$$

which concludes the proof for $p < j$.

B.5.1 Proof of Lemma 14

We obtain Lemma 14 by re-writing the terms inside $a_\tau^{(p)}$ such that some of the terms do not impact the optimization problem, and hence we can remove them.

Obtaining a new expression for $a_\tau^{(p)}$

It follows that, for every $\tau \in [\Lambda]$, $a_\tau^{(p)}$ defined in (B.53) can be written as

$$a_\tau^{(p)} = \sum_{q\in C_{p+1}^{[\Lambda]}} \dot{\boldsymbol{L}}_q \frac{1}{j+1-|q\cap\tau|} = \sum_{m=0}^{\min(j,p+1)} \frac{1}{j+1-m} \sum_{\substack{q\in C_{p+1}^{[\Lambda]}\\ |q\cap\tau|=m}} \dot{\boldsymbol{L}}_q, \tag{B.63}$$

where the $\min(j, p+1)$ term in the summation comes from the fact that $|q \cap \tau| \leq \min(|q|, |\tau|)$. For a given τ, let η be a subset of τ of cardinality m, such that $\eta \subseteq \tau$, $|\eta| = m$. The last sum in (B.63) can be expanded as

$$\sum_{\substack{q\in C_{p+1}^{[\Lambda]}\\ |q\cap\tau|=m}} \dot{\boldsymbol{L}}_q = \sum_{\substack{\eta\subseteq\tau\\ |\eta|=m}} \sum_{\substack{q\in C_{p+1}^{[\Lambda]}\\ q\cap\tau=\eta}} \dot{\boldsymbol{L}}_q. \tag{B.64}$$

Let us consider a particular $\eta \subseteq \tau$, $|\eta| = m$. Then,

$$\sum_{\substack{q\in C_{p+1}^{[\Lambda]}\\ q\cap\tau=\eta}} \dot{\boldsymbol{L}}_q = \sum_{\substack{q\in C_{p+1}^{[\Lambda]\backslash\{\tau\backslash\eta\}}\\ \eta\subseteq q}} \dot{\boldsymbol{L}}_q = \dot{\boldsymbol{L}}_\eta \sum_{q\in C_{p+1-m}^{[\Lambda]\backslash\{\tau\}}} \dot{\boldsymbol{L}}_q. \tag{B.65}$$

Let us recall that the term $\sum_{q\in C_{p+1}^{[\Lambda]}} \dot{\boldsymbol{L}}_q$ represents the $(p+1)$-th elementary symmetric polynomial for the set $\mathbf{L} \triangleq \{L_\lambda\}_{\lambda=1}^{\Lambda}$, and that the elementary symmetric polynomials satisfy Property 1. Hence, for any set of positive integers Ω and any integer n such that $n \notin \Omega$, and by making use of the notation defined above, we can rewrite Property 1 as

$$L_n \sum_{q\in C_p^\Omega} \dot{\boldsymbol{L}}_q = \sum_{q\in C_{p+1}^{\{\Omega\cup n\}}} \dot{\boldsymbol{L}}_q - \sum_{q\in C_{p+1}^{\Omega}} \dot{\boldsymbol{L}}_q. \tag{B.66}$$

This is equivalent to say that the sum over the $\dot{\boldsymbol{L}}_q$ ($|q| = p+1$) that include L_n is equal to the sum over all the $\dot{\boldsymbol{L}}_q$ ($|q| = p+1$) minus the sum over the $\dot{\boldsymbol{L}}_q$ ($|q| = p+1$) that does not include L_n. This intuitive relation will prove fundamental for the derivation.

Note that $\dot{\boldsymbol{L}}_\eta = \prod_{i\in\eta} L_i$. Then, we can iteratively apply (B.66) to (B.65) for all the L_i, $i \in \eta$, such that (B.65) is expanded in 2^m sums as

$$\sum_{\substack{q\in C_{p+1}^{[\Lambda]}\\ q\cap\tau=\eta}} \dot{\boldsymbol{L}}_q = \sum_{\kappa\subseteq\eta}(-1)^{|\kappa|} \sum_{q\in C_{p+1}^{[\Lambda]\backslash\{\tau\backslash\{\eta\backslash s\}\}}} \dot{\boldsymbol{L}}_q = \sum_{k=0}^{m}(-1)^k \sum_{\substack{\kappa\subseteq\eta\\ |\kappa|=k}} \sum_{q\in C_{p+1}^{[\Lambda]\backslash\{\tau\backslash\{\eta\backslash\kappa\}\}}} \dot{\boldsymbol{L}}_q. \tag{B.67}$$

Thus, from (B.63), (B.64), and (B.67), we obtain that

$$a_\tau^{(p)} = \sum_{m=0}^{\min(j,p+1)} \frac{1}{j+1-m} \sum_{\substack{\eta\subseteq\tau\\ |\eta|=m}} \sum_{k=0}^{m}(-1)^k \sum_{\substack{\kappa\subseteq\eta\\ |\kappa|=k}} \sum_{q\in C_{p+1}^{[\Lambda]\backslash\{\tau\backslash\{\eta\backslash\kappa\}\}}} \dot{\boldsymbol{L}}_q. \tag{B.68}$$

Since $\{\tau\backslash\{\eta\backslash\kappa\}\}$ can be the same set for different η, κ, let us count how many times the term $\sum_{q\in C_{p+1}^{[\Lambda]\backslash\{w\}}} \dot{\boldsymbol{L}}_q$ appears in (B.68) for a certain $w \subseteq [\tau]$, $|w| = i$. Let us fix m (i.e., the cardinality of $|\eta|$) and $j = |\tau|$. It follows that $|w| = |\{\tau\backslash\{\eta\backslash\kappa\}\}| = j - (m-k)$. We have that $w = \tau\backslash\{\eta\backslash\kappa\} = \{\tau\backslash\eta\} \cup \kappa$. This implies that $\kappa \subseteq w$. Furthermore, for any $\kappa \subseteq w$, there exists a distinct and unique η such that $w = \{\tau\backslash\eta\} \cup \kappa$. Since there are $\binom{|w|}{|\kappa|} = \binom{i}{k} = \binom{i}{i+m-j}$ possible $\kappa \subseteq w$ of cardinality k, each of the terms $\sum_{q\in C_{p+1}^{[\Lambda]\backslash\{w\}}} \dot{\boldsymbol{L}}_q$ appears in (B.68) exactly $\binom{i}{i+m-j}$ times for a particular m, i and j.

Let us denote the sum over all $q \in C_{p+1}^{[\Lambda]\backslash\{w\}}$ for any w of cardinality $|w| = i$ as $\Theta_{\backslash i}$, such that

$$\Theta_{\backslash i} \triangleq \sum_{\substack{w\subseteq\tau\\ |w|=i}} \sum_{q\in C_{p+1}^{[\Lambda]\backslash\{w\}}} \dot{\boldsymbol{L}}_q. \tag{B.69}$$

From (B.69) and the fact that the term $\sum_{q\in C_{p+1}^{[\Lambda]\backslash\{w\}}} \dot{\boldsymbol{L}}_q$ appears in (B.68) exactly $\binom{i}{i+m-j}$ times for a particular m, i and j, and after applying $k = i+m-j$, it follows that

$$\sum_{k=0}^{m}(-1)^k \sum_{\substack{\eta\subseteq\tau\\ |\eta|=m}} \sum_{\substack{\kappa\subseteq\eta\\ |\kappa|=k}} \sum_{q\in C_{p+1}^{[\Lambda]\backslash\{\tau\backslash\{\eta\backslash\kappa\}\}}} \dot{\boldsymbol{L}}_q = \sum_{i=j-m}^{j} (-1)^{i+m-j} \binom{i}{i+m-j} \Theta_{\backslash i}, \tag{B.70}$$

where we have substituted $k = i+m-j$.

Then, we can apply (B.70) into (B.68) to obtain that

$$a_\tau^{(p)} = \sum_{m=0}^{\min(j,p+1)} \sum_{i=j-m}^{j} \frac{1}{j+1-m}(-1)^{i+m-j} \binom{i}{i+m-j} \Theta_{\backslash i}. \tag{B.71}$$

This expression of $a_\tau^{(p)}$ can be further simplified. Before continuing, let us take a look at the term $\Theta_{\backslash i}$. We show in the following that not all the components of $\Theta_{\backslash i}$ will impact the optimization of $a_\tau^{(p)}$.

Reducing $\Theta_{\backslash i}$ to its meaningful components

We recall that, as expressed in (B.55), our initial goal is to consider only the last element of the set τ, i.e., τ_j, for any fixed set $\tau_{[j-1]} = \{\tau_1, \ldots, \tau_{j-1}\}$, and obtain

$$\underset{\tau_j \in \{[\Lambda] \backslash \tau_{[j-1]}\}}{\operatorname{argmin}} \operatorname{sgn}(p-j) a_{\tau}^{(p)}. \tag{B.72}$$

Note that the cache groups are ordered such that $L_1 \geq \cdots \geq L_\Lambda$. For a given set $\tau_{[j-1]} \subset [\Lambda]$ of cardinality $j-1$, we have to find the cache in the remaining set $[\Lambda] \backslash \tau_{[j-1]}$ that optimizes (B.72), which we denote by τ_j.

Note that $\Theta_{\backslash i} \triangleq \sum_{\substack{w \subseteq \tau \\ |w|=i}} \sum_{q \in C_{p+1}^{[\Lambda]\backslash\{w\}}} \dot{\boldsymbol{L}}_q$, defined in (B.67), is composed of $\binom{j}{i}$ sums, one for each $w \subseteq \tau : |w| = i$. Interestingly, if $w \subseteq \tau$ is actually a subset of $\tau_{[j-1]}$ ($w \subseteq \tau_{[j-1]}$), the term $\sum_{q \in C_{p+1}^{[\Lambda]\backslash\{w\}}} \dot{\boldsymbol{L}}_q$ is the same no matter which value in $\{[\Lambda] \backslash \tau_{[j-1]}\}$ is selected as τ_j. Thus, such terms are irrelevant for the optimization problem.

Let us denote the $\binom{j-1}{i-1}$ subsets w which contain τ_j in (B.67) as $\Theta_{\backslash i-1}^{\backslash \tau_j}$, i.e.,

$$\Theta_{\backslash i-1}^{\backslash \tau_j} \triangleq \sum_{\substack{\nu \subseteq \{\tau \backslash \tau_j\} \\ |\nu|=i-1}} \sum_{q \in C_{p+1}^{[\Lambda]\backslash\{\tau_j,\nu\}}} \dot{\boldsymbol{L}}_q, \tag{B.73}$$

and define

$$b_{\tau}^{(p)} \triangleq \sum_{m=0}^{\min(j,p+1)} \sum_{i=j-m}^{j} \frac{1}{j+1-m} (-1)^{i+m-j} \binom{i}{i+m-j} \Theta_{\backslash i-1}^{\backslash \tau_j}, \tag{B.74}$$

where $b_{\tau}^{(p)}$ is obtained by substituting $\Theta_{\backslash i}$ in a_{τ}^{p} by $\Theta_{\backslash i-1}^{\backslash \tau_j}$. Then, it follows that

$$\underset{\tau_j \in \{[\Lambda] \backslash \tau_{[j-1]}\}}{\operatorname{argmin}} \operatorname{sgn}(p-j) a_{\tau}^{(p)} = \underset{\tau_j \in \{[\Lambda] \backslash \tau_{[j-1]}\}}{\operatorname{argmin}} \operatorname{sgn}(p-j) b_{\tau}^{(p)}. \tag{B.75}$$

Next, we simplify $\Theta_{\backslash i-1}^{\backslash \tau_j}$ to later apply this result into $b_{\tau}^{(p)}$ and obtain Lemma 14.

Simplifying the term $\Theta_{\backslash i-1}^{\backslash \tau_j}$

The elements in a combination of $p+1$ elements in $[\Lambda] \backslash \{\tau_j, \nu\}$ can be expressed as the concatenation of ℓ elements of $\{\tau \backslash \{\tau_j, \nu\}\}$ and $p+1-\ell$ elements of $\big\{[\Lambda] \backslash \{\tau_j, \nu\}\big\} \backslash \big\{\tau \backslash \{\tau_j, \nu\}\big\}$ $= [\Lambda] \backslash \{\tau\}$, for any $\ell \in [j-i]_0$. Consequently, it follows that

$$\sum_{q \in C_{p+1}^{[\Lambda]\backslash\{\tau_j,\nu\}}} \dot{\boldsymbol{L}}_q = \sum_{\ell=0}^{j-i} \sum_{\substack{\mu \subseteq \{\tau \backslash \{\tau_j,\nu\}\} \\ |\mu|=\ell}} \dot{\boldsymbol{L}}_\mu \sum_{q \in C_{p+1-\ell}^{[\Lambda]\backslash\{\tau\}}} \dot{\boldsymbol{L}}_q. \tag{B.76}$$

Applying (B.76) into (B.73) yields

$$\Theta_{\backslash i-1}^{\backslash \tau_j} = \sum_{\substack{\nu \subseteq \{\tau \backslash \tau_j\} \\ |\nu|=i-1}} \sum_{\ell=0}^{j-i} \sum_{\substack{\mu \subseteq \{\tau \backslash \{\tau_j, \nu\}\} \\ |\mu|=\ell}} \dot{\boldsymbol{L}}_\mu \sum_{q \in C_{p+1-\ell}^{[\Lambda]\backslash\{\tau\}}} \dot{\boldsymbol{L}}_q \tag{B.77}$$

$$= \sum_{\ell=0}^{j-i} \Big(\underbrace{\sum_{\substack{\nu \subseteq \{\tau \backslash \tau_j\} \\ |\nu|=i-1}} \sum_{\substack{\mu \subseteq \{\tau \backslash \{\tau_j, \nu\}\} \\ |\mu|=\ell}} \dot{\boldsymbol{L}}_\mu}_{E_{j-1,\ell}} \Big) \sum_{q \in C_{p+1-\ell}^{[\Lambda]\backslash\{\tau\}}} \dot{\boldsymbol{L}}_q. \tag{B.78}$$

Next, we want to count how many times the last sum ($\sum_{q \in C_{p+1-\ell}^{[\Lambda]\backslash\{\tau\}}} \dot{\boldsymbol{L}}_q$) appears in $\Theta_{\backslash i-1}^{\backslash \tau_j}$. Consider some given i and ℓ. In the term $E_{j-1,\ell}$ in (B.78), a specific $\dot{\boldsymbol{L}}_\mu$ appears $\binom{j-1-\ell}{i-1}$ times. Then, it holds that

$$\Theta_{\backslash i-1}^{\backslash \tau_j} = \sum_{\ell=0}^{j-i} \binom{j-1-\ell}{i-1} \sum_{\substack{\mu \subseteq \{\tau \backslash \tau_j\} \\ |\mu|=\ell}} \dot{\boldsymbol{L}}_\mu \sum_{q \in C_{p+1-\ell}^{[\Lambda]\backslash\{\tau\}}} \dot{\boldsymbol{L}}_q. \tag{B.79}$$

Now, we substitute $\Theta_{\backslash i-1}^{\backslash \tau_j}$ by (B.79) in (B.74) to obtain Lemma 14.

Obtaining (B.58)

Let us introduce the notation $\varpi \triangleq \min(j, p+1)$ for ease of readability. Then,

$$b_\tau^{(p)} \triangleq \sum_{m=0}^{\varpi} \sum_{i=j-m}^{j} \frac{1}{j+1-m} (-1)^{i+m-j} \binom{i}{i+m-j} \Theta_{\backslash i-1}^{\backslash \tau_j} \tag{B.80}$$

$$\overset{(a)}{=} \sum_{i=j-\varpi}^{j} \sum_{m'=j-\varpi}^{i} \frac{1}{m'+1} (-1)^{i-m'} \binom{i}{i-m'} \Theta_{\backslash i-1}^{\backslash \tau_j} \tag{B.81}$$

$$\overset{(b)}{=} \sum_{i=j-\varpi}^{j} \frac{(-1)^{i-j+\varpi}}{i+1} \binom{i}{j-\varpi} \Theta_{\backslash i-1}^{\backslash \tau_j}, \tag{B.82}$$

where (a) follows from interchanging the summations and applying the change of variable $m' = j - m$, and in (b) we have solved the inner summation.

Let us now substitute $\Theta_{\backslash i-1}^{\backslash \tau_j}$ by (B.79) in (B.82), which leads to

$$b_\tau^{(p)} = \sum_{i=j-\varpi}^{j} \frac{(-1)^{i-j+\varpi}}{i+1} \binom{i}{j-\varpi} \sum_{\ell=0}^{j-i} \binom{j-1-\ell}{i-1} \sum_{\substack{\mu \subseteq \{\tau \backslash \tau_j\} \\ |\mu|=\ell}} \dot{\boldsymbol{L}}_\mu \sum_{q \in C_{p+1-\ell}^{[\Lambda]\backslash\{\tau\}}} \dot{\boldsymbol{L}}_q. \tag{B.83}$$

By interchanging the summations (since $\sum_{i=j-\varpi}^{j}\sum_{\ell=0}^{j-i} f(i,\ell) = \sum_{\ell=0}^{\varpi}\sum_{i=j-\varpi}^{j-\ell} f(i,\ell)$), we have that

$$b_{\tau}^{(p)} = \sum_{\ell=0}^{\varpi}\sum_{i=j-\varpi}^{j-\ell} F_{\ell,p,j} \sum_{\substack{\mu\subseteq\{\tau\setminus\tau_j\}\\|\mu|=\ell}} \dot{\boldsymbol{L}}_{\mu} \sum_{q\in C_{p+1-\ell}^{[\Lambda]\setminus\{\tau\}}} \dot{\boldsymbol{L}}_{q}, \tag{B.84}$$

where $F_{\ell,p,j} \triangleq \frac{(-1)^{i-j+\varpi}}{i+1}\binom{i}{j-\varpi}\binom{j-1-\ell}{i-1}$. Let us consider $\sum_{i=j-\varpi}^{j-\ell} F_{\ell,p,j}$. To simplify the notation, let us define $h \triangleq j-\varpi$ and $g = \varpi - \ell$. Thus, it follows that

$$\sum_{i=j-\varpi}^{j-\ell} F_{\ell,p,j} = \sum_{i=h}^{h+g} \frac{(-1)^{i-h}}{i+1}\binom{i}{h}\binom{h+g-1}{i-1} \tag{B.85}$$

$$= \frac{-1}{(h+g+1)(h+g)} = \frac{-1}{(j-\ell)(j-\ell+1)}. \tag{B.86}$$

Thus, we obtain that

$$b_{\tau}^{(p)} = \sum_{\ell=0}^{\varpi} \frac{-1}{(j-\ell)(j-\ell+1)} \sum_{\substack{\mu\subseteq\{\tau\setminus\tau_j\}\\|\mu|=\ell}} \dot{\boldsymbol{L}}_{\mu} \sum_{q\in C_{p+1-\ell}^{[\Lambda]\setminus\{\tau\}}} \dot{\boldsymbol{L}}_{q}, \tag{B.87}$$

which concludes the proof of (B.58) and Lemma 14. □

B.6 Proof of Theorem 6

In order to prove Theorem 6, we have to prove that the achievable delivery time in (4.55) matches the lower bound in (4.60). To do so, we first notice that (4.55) and (4.60) have the same denominator, thus leaving us to prove that the numerator of the achievable delivery time is exactly equal to the numerator of the bound, i.e.,

$$\sum_{\tau\in C_{t+1}^{[\Lambda]}} \max_{\lambda\in\tau} L'_{\lambda}\dot{\boldsymbol{L}}_{\tau\setminus\lambda} = \max_{\sigma\in S_{\Lambda,\Lambda-t}} \sum_{\lambda=1}^{\Lambda-t} L'_{\sigma(\lambda)} \sum_{\tau\in C_{t}^{[\Lambda]\setminus\{\sigma(1),\ldots,\sigma(\lambda)\}}} \dot{\boldsymbol{L}}_{\tau}. \tag{B.88}$$

We start by constructing the set

$$\Phi \triangleq \left\{ \max_{i\in\tau} L'_i \dot{\boldsymbol{L}}_{\tau\setminus\{i\}} : \tau\subset[\Lambda], |\tau| = t+1 \right\}, \tag{B.89}$$

which is comprised of the addends of the left-hand-side of (B.88). We naturally have that $|\Phi| = \binom{\Lambda}{t+1}$.

In the following, we make use of the term *leader* in a specific way. In particular, we say that j is a leader of a set $\tau\in[\Lambda]$ if $j = \operatorname{argmax}_{i\in\tau} L'_i\dot{\boldsymbol{L}}_{\tau\setminus\{i\}}$. The next lemma shows that the caches that act as leaders follow a particular structure.

Lemma 15. *For any $j \in [\Lambda]$, let us denote the number of times that j is a leader in Φ as $\mathcal{N}_j$. Then, $\mathcal{N}_j$ satisfies that*

$$\mathcal{N}_j \in \left\{0, \binom{t}{t}, \binom{t+1}{t}, \dots, \binom{\Lambda-1}{t}\right\}. \tag{B.90}$$

Proof. Without loss of generality, let us assume that j is such that $L_j' L_i \geq L_i' L_j \ \forall i \in \phi$ for some $\phi \subset [\Lambda] \setminus \{j\}, |\phi| = m$, and for some $m \in \{t, t+1, \dots, \Lambda-1\}$. Notice that we must have $m \geq t$, since, for $m < t$, j cannot be a leader in Φ. The fact that there exist $\binom{m}{t}$ $t-$combinations of ϕ implies that j is a leader in Φ at least $\binom{m}{t}$ times, since $L_j' L_\tau \geq L_i' L_{\{j\}\cup\tau\setminus\{i\}}, \forall i \in \tau, \forall \tau \in C_t^\phi$ by assumption. This and the fact that j might not be a leader in Φ completes the proof. □

Let us now consider the set of all the leaders $\ell_1, \ell_2, \ell_3, \dots, \ell_\Lambda$ in Φ, and let us sort them such that, without loss of generality, we assume that $L_{\ell_j}' L_{\ell_{j+1}} \geq L_{\ell_{j+1}}' L_{\ell_j}$ for any $j \in [\Lambda - 1]$. It can be easily verified that this order of the leaders implies

$$L_{\ell_j}' L_{\ell_i} \geq L_{\ell_i}' L_{\ell_j} \qquad \forall i > j. \tag{B.91}$$

Let us consider $j = 1$. From the above, we have that $L_{\ell_1}' L_{\ell_i} \geq L_{\ell_i}' L_{\ell_1}$ for any $i > 1$. We can multiply both sides of the inequality by $\dot{\boldsymbol{L}}_\eta$ for any $\eta \subseteq [\Lambda] \setminus \{\ell_1, \ell_i\}$ of cardinality $|\eta| = t-1$. There are $\binom{\Lambda-2}{t-1}$ such subsets for each i, thus a total of $(\Lambda-1)\binom{\Lambda-2}{t-1}$ different inequalities. This writes as

$$L_{\ell_1}' L_{\ell_i} \dot{\boldsymbol{L}}_\eta \geq L_{\ell_i}' L_{\ell_1} \dot{\boldsymbol{L}}_\eta \qquad \forall \eta \subseteq [\Lambda] \setminus \{\ell_1, \ell_i\}, \quad \forall i > 1. \tag{B.92}$$

For ℓ_1 to be the leader of a set τ, $|\tau| = t+1$, we need $L_{\ell_1}' L_{\ell_i} \dot{\boldsymbol{L}}_{\tau\setminus\{\ell_1,\ell_i\}} \geq L_{\ell_i}' L_{\ell_1} \dot{\boldsymbol{L}}_{\tau\setminus\{\ell_1,\ell_i\}}$ for any $i : \ell_i \in \tau \setminus \ell_1$. That is, we need t different inequalities among those in (B.92), each one from a different i. Then, the set of $(\Lambda-1)\binom{\Lambda-2}{t-1}$ different inequalities in (B.92) imply that ℓ_1 is a leader of $\frac{\Lambda-1}{t}\binom{\Lambda-2}{t-1} = \binom{\Lambda-1}{t}$ different sets $\tau \subseteq [\Lambda]$, $|\tau| = t+1$. Indeed, that amounts to all the possible sets in which ℓ_1 appears.

After considering ℓ_1 for the sake of comprehension, let us consider a general ℓ_j. Let us now multiply (B.91) by $\dot{\boldsymbol{L}}_\eta$ for any $\eta \subseteq [\Lambda] \setminus \{\{\ell_k\}_{k\leq j}, \ell_i\}$ of cardinality $|\eta| = t-1$. Note that now we have only considered the subsets that do not include neither ℓ_i nor any ℓ_k for $k \leq j$. Hence, we have $\binom{\Lambda-j-1}{t-1}$ such subsets for each i, thus a total of $(\Lambda-j)\binom{\Lambda-j-1}{t-1}$ different inequalities.

Now, let us denote by Ω_j the set of subsets of cardinality $t+1$ which contain ℓ_j but do not contain any ℓ_k such that $k < j$, i.e., $\Omega_j \triangleq \{\tau : |\tau| = t+1, \ \tau \ni j, \{\tau\setminus\ell_j\} \subseteq [\Lambda]\setminus\{\ell_k\}_{k\leq j}\}$. Note that within the previous set of inequalities, i.e., within

$$L_{\ell_j}' L_{\ell_i} \dot{\boldsymbol{L}}_\eta \geq L_{\ell_i}' L_{\ell_j} \dot{\boldsymbol{L}}_\eta \qquad \forall \eta \subseteq [\Lambda] \setminus \{\{\ell_k\}_{k\leq j}, i\}, \quad \forall i > j, \tag{B.93}$$

we can find the t inequalities required for ℓ_j to be the leader of any subset τ in Ω_j (since, for an arbitrary $\tau \in \Omega_j$, we need $L_{\ell_j}' L_{\ell_i} \dot{\boldsymbol{L}}_{\tau\setminus\{\ell_j,\ell_i\}} \geq L_{\ell_i}' L_{\ell_j} \dot{\boldsymbol{L}}_{\tau\setminus\{\ell_j,\ell_i\}}$ for any $i : \ell_i \in \tau \setminus \ell_j$). Consequently, the set of $(\Lambda-j)\binom{\Lambda-j-1}{t-1}$ different inequalities in (B.93) imply that ℓ_j is a

leader of $\frac{\Lambda-j}{t}\binom{\Lambda-j-1}{t-1} = \binom{\Lambda-j}{t}$ different sets $\tau \subseteq [\Lambda]$, $|\tau| = t+1$. Indeed, that amounts to all the possible sets in which ℓ_j appears and no ℓ_k, $k < j$, appears.

Interestingly, this implies that each ℓ_j is the leader of all the sets τ in which it appears *and* none of the previous $\{\ell_k\}_{k<j}$ appears. Summing up all the sets for which $\ell_1, \dots, \ell_{\Lambda-t}$ are leaders yields

$$\sum_{n=1}^{\Lambda-t} \binom{\Lambda-n}{t} = \binom{\Lambda}{t+1}, \tag{B.94}$$

which matches the cardinality of Φ. Hence, there are only $\Lambda - t$ leaders.[1]

Finally, by considering all possible $(\Lambda-t)$-combinations of $[\Lambda]$ as all the possible set of leaders, we can conclude that

$$\sum_{q \in C_{t+1}^{[\Lambda]}} \max_{\lambda \in q} L'_\lambda \dot{\boldsymbol{L}}_{q\setminus\lambda} = \max_{\sigma \in S_{\Lambda,\Lambda-t}} \sum_{\lambda=1}^{\Lambda-t} L'_{\sigma(\lambda)} \sum_{q \in C_t^{[\Lambda]\setminus\{\sigma(1),\dots,\sigma(\lambda)\}}} \dot{\boldsymbol{L}}_q, \tag{B.95}$$

which concludes the proof. □

[1] Note that, for $\ell_{\Lambda-t+1}$, the number of possible subsets of cardinality $t+1$ in $[\Lambda]$ which do not contain any element in $\{\ell_k\}_{k\in[\Lambda-t+1]}$ is zero because the set $[\Lambda] \setminus \{\ell_k\}_{k\in[\Lambda-t]}$ has cardinality t.

Appendix C

Proofs of Chapter 6

C.1 More Detailed Analysis of the Delivery Phase

For the delivery and decoding procedures presented in Sections 6.3.2 and 6.3.3, we here show that at the end of the K rounds, each user successfully decodes all its missing data parts. We recall that for the placement matrix $\mathbf{V}$, if $\mathbf{V}[p,k] = 0$ for some $p \in [K]$ and $k \in [K]$, then the subfile $W_p^{(d_k)}$, which is comprised of $t+\alpha$ smaller minifiles of the form $W_{p,q}^{(d_k)}$, must be delivered to user k. By construction of the user index and packet index vectors $\mathbf{k}_j^r$ and $\mathbf{p}_j^r$, where $r \in [K]$ and $j \in [t+\alpha]$, it is easy to see that for any $p \in [K]$ and $k \in [K]$, if $\mathbf{V}[p,k] = 0$ there exists always a transmission vector that contains one of the $t+\alpha$ subpackets represented by $\mathbf{V}[p,k]$; i.e. there exists always a triple (j,r,n) such that $p = \mathbf{p}_j^r[n]$ and $k = \mathbf{k}_j^r[n]$. Therefore, it is sufficient to show that if $\mathbf{V}[p,k] = 0$ for some pair (p,k), there exist exactly $t+\alpha$ triples (j,r,n) for which $p = \mathbf{p}_j^r[n]$ and $k = \mathbf{k}_j^r[n]$.

Without loss of generality, let us consider the first round of the delivery scheme. The vector $\mathbf{k}_j^1$ contains the index of the users targeted at transmission j of this round. Let us split the user indices in $\mathbf{k}_j^1$ into the following two vectors

$$\mathbf{c}_j^1 = [1:t]\ , \quad \mathbf{s}_j^1 = [(([1:\alpha]+j-1)\%(K-t))+t]\ . \tag{C.1}$$

Similarly, we split the packet indices in $\mathbf{p}_j^1$ into the following two vectors

$$\mathbf{r}_j^1 = [((t+j-[1:t])\%(K-t))+[1:t]]\ , \quad \mathbf{d}_j^1 = \mathbf{e}(\alpha)\ .$$

Now let us consider the set of points in the tabular representation corresponding to the vector pair $(\mathbf{r}_1^j, \mathbf{c}_1^j)$; i.e., the set of points in which the row index is taken from $\mathbf{r}_1^j$ and column index is taken from $\mathbf{c}_1^j$. We will use $\{(\mathbf{r}_1^j, \mathbf{c}_1^j)\}$ to denote this set. From the graphical example in Section 6.3.4, we recall that $\{(\mathbf{r}_1^j, \mathbf{c}_1^j)\}$ is an element-wise circular shift of $\{(\mathbf{r}_1^1, \mathbf{c}_1^1)\}$, over the non-shaded cells of the tabular representation and in the vertical direction. Similarly, $\{(\mathbf{d}_1^j, \mathbf{s}_1^j)\}$ is an element-wise circular shift of $\{(\mathbf{d}_1^1, \mathbf{s}_1^1)\}$, over the non-shaded cells and in the horizontal direction. As a result, in round 1, the following two statements hold:

- for every user $k \in [K]$ and packet index $p \in [K]$, if k is available in $\mathbf{c}_1^1 = \mathbf{c}_2^1 = ... = \mathbf{c}_{K-t}^1$ and p is available in $[\mathbf{r}_1^1 \| ... \| \mathbf{r}_{K-t}^1]$, there exists exactly one transmission index $j \in [K-t]$, such that the j-th transmission of the first round delivers $W_{p,q}^{(d_k)}$ to user k, for some subpacket index $q \in [t+\alpha]$. In other words there exists exactly one triple $(j, 1, n)$ for the pair (p, k);

- for every user $k \in [K]$, if k is available in $[\mathbf{s}_1^1 \| \mathbf{s}_2^1 \| ... \| \mathbf{s}_{K-t}^1]$, there exist exactly α transmissions in the first round that deliver $W_{1,q}^{(d_k)}$ to user k, each with a distinct subpacket index $q \in [t+\alpha]$. In other words, there exist exactly α triples $(j, 1, n)$, for the pair $(1, k)$.

These statements also hold for other transmission rounds, i.e., transmission round r where $1 < r \leq K$. However, for any $k, p \in [K]$ such that $\mathbf{V}[p, k] = 0$, there exists exactly one round r for which k appears in $[\mathbf{s}_1^r \| \mathbf{s}_2^r \| ... \| \mathbf{s}_{K-t}^r]$, while there exist t different rounds r for which k is available in $\mathbf{c}_1^r = \mathbf{c}_2^r = ... = \mathbf{c}_{K-t}^r$. Hence, for any $k, p \in [K]$ such that $\mathbf{V}[p, k] = 0$, there exist $\alpha \times 1 + 1 \times t = \alpha + t$ triples (j, r, n) for the pair (p, k). This clarifies the correctness of the delivery algorithm proposed in Section 6.3.2.

C.2 Reducing Subpacketization by a Factor of $\phi_{K,t,\alpha}^2$

To reduce the subpacketization requirement, we apply a user grouping mechanism, inspired by [33]. For notation simplicity, let us use simply use ϕ to represent $\phi_{K,t,\alpha}$. The idea is to split users into groups of size $\phi_{K,t,\alpha}$ and assume each group is equivalent to a *virtual* user. Then, we consider a virtual network consisting of these virtual users, in which the coded caching and spatial multiplexing gains are $\frac{t}{\phi}$ and $\frac{\alpha}{\phi}$, respectively. Finally, we apply the coded caching scheme proposed in Sections 6.3.1 and 6.3.2 to the virtual network, and *elevate* the resulting cache placement and delivery schemes to be applicable in the original network. Here, we provide a detailed description of the elevation procedure.

Cache Placement

Assume the original network is given with K users, coded caching gain of t, and spatial DoF of α. We first split the set of users $[K]$ into $K' = \frac{K}{\phi}$ disjoints groups $v_{k'}$, $k' \in [K']$, where all groups have the same number of ϕ users. Without loss of generality, we assume each group $v_{k'}$ corresponds to the set of users

$$v_{k'} \triangleq [\phi * (k'-1) + 1 : \phi * k'] \ , \ \ \forall k' \in [K'] \ . \tag{C.2}$$

Next, we assume each user group is equivalent to a virtual user with cache size of Mf bits, and the virtual users form a new virtual network in which the spatial DoF is $\alpha' = \frac{\alpha}{\phi}$. For this virtual network, we use the same cache placement algorithm presented in Section 6.3.1, and set the cache content of every user in group $v_{k'}$ to be the same as the cache content of the virtual user corresponding to $v_{k'}$. The total subpacketization is then $K'(t' + \alpha')$, where $t' = \frac{t}{\phi}$.

Delivery Phase

During the delivery phase, we first create transmission vectors for the virtual network, and then *elevate* them to be applicable in the original network. Following (6.2), the transmission vector i for the virtual network is built as $\mathbf{x}'_i = \sum_{v_{k'} \in \mathcal{X}'_i} \mathbf{w}'_i(v_{k'}) X'_i(v_{k'})$, in which $\mathcal{X}'_i$ is the set of virtual users targeted at transmission i, $X'_i(v_{k'})$ denotes the data part targeted to the virtual user $v_{k'}$ during the same transmission, and $\mathbf{w}'_i(v_{k'})$ is the beamformer vector assigned to $X'_i(v_{k'})$. In order to elevate $\mathbf{x}'_i$ for the original network, we first notice than each virtual user represents a set of ϕ original users; and hence, using (C.2), the set of targeted users during transmission i for the original network is

$$\mathcal{X}_i = \bigcup_{v_{k'} \in \mathcal{X}'_i} [\phi * (k'-1) + 1 : \phi * k'] \, . \tag{C.3}$$

Following the discussions in Section 6.3.5, every beamformer vector $\mathbf{w}'_i(v_{k'})$ is built to suppress unwanted terms at $\alpha' - 1$ virtual users. Let us denote the set of such virtual users as $\mathcal{R}'_i(v_{k'})$, where $|\mathcal{R}'_i(v_{k'})| = \alpha' - 1$. The goal is then to find the respective set for the original network, denoted by $\mathcal{R}_i(k)$, for $k \in [K]$. As the spatial DoF for the original network is α, it is possible to suppress undesired terms at $\alpha - 1$ original users; i.e. $|\mathcal{R}_i(k)| = \alpha - 1$. Without loss of generality, let us assume that user k is in the respective group of $v_{k'}$ (every user in the original network has one counterpart in the virtual network). In the original network, for the interference to be suppressed, the following conditions should be met:

1. For every $v_{\hat{k}'} \in \mathcal{R}'_i(v_{k'})$, $\mathcal{R}_i(k)$ should include all the users in the respective group of $v_{\hat{k}'}$; i.e. $\mathcal{R}_i(k)$ should include all the users in $[\phi * (\hat{k}' - 1) + 1 : \phi * \hat{k}']$;
2. $\mathcal{R}_i(k)$ should include all other users in the respective group of $v_{k'}$; i.e. $\mathcal{R}_i(k)$ should include all the users in $[\phi * (k'-1) + 1 : \phi * k'] \backslash \{k\}$.

Using a formal representation, we have

$$\mathcal{R}_i(k) = [\phi * (k'-1) + 1 : \phi * k'] \backslash \{k\} \bigcup_{v_{\hat{k}'} \in \mathcal{R}'_i(v_{k'})} [\phi * (\hat{k}' - 1) + 1 : \phi * \hat{k}'] \, . \tag{C.4}$$

In other words, the data part $X_i(k)$ intended for user k at transmission i has to be suppressed not only at $\alpha' - 1$ virtual users (where each virtual user represents ϕ original users), but also at $\phi - 1$ original users in the equivalent group of $v_{k'}$. Hence, the total number of users for which $X_i(k)$ should be suppressed is $(\alpha' - 1)\phi + \phi - 1$. Substituting $\alpha' = \frac{\alpha}{\phi}$, we have

$$|\mathcal{R}_i(k)| = (\frac{\alpha}{\phi} - 1)\phi + \phi - 1 = \alpha - 1 \, . \tag{C.5}$$

Now, we can elevate $\mathbf{x}'_i$ to be applicable in the original network. All we need to do is to use (C.3) to substitute the target user set $\mathcal{X}'_i$ with $\mathcal{X}_i$, and replace $\mathbf{w}'_i(v_{k'}) X'_i(v_{k'})$ with

$$\sum_{k \in [\phi * (k'-1) + 1 : \phi * k']} \mathbf{w}_i(k) X_i(k) \, , \tag{C.6}$$

where $\mathbf{w}_i(k)$ is the beamformer vector designed to suppress unwanted data terms at the user set $\mathcal{R}_i(k)$ as defined in (C.4), and $X_i(k)$ is the data part intended for user k at transmission i. The subpacketization for the virtual network, $K'(t' + \alpha')$, would then be still valid for the original network, indicating a reduction of ϕ^2 compared with the case when no grouping is applied.

Appendix D

Appendix of Chapter 8

D.1 Proof of Lemma 12

We prove the lemma by induction as in [16]. To simplify the notation, in the rest of the proof we will drop the index $\boldsymbol{\mathcal{M}}$ whenever present.

First, we prove the statement for $S = \{\lambda\}$, for any $\lambda \in [\Lambda]$. We have that

$$H(X_\lambda | Y_{[\Lambda]\setminus\{\lambda\}}) \geq 0 = T a_\lambda^{(\lambda,1)} \frac{L_\lambda - 1 \cdot L_\lambda}{1^2} \tag{D.1}$$

Now, we suppose the statement in Lemma 12 is true for all subsets of size S_0. Combining equations (48), (55) and (58) in [16], for any $S \subseteq \{1, 2, \ldots, \Lambda\}$ of size $|S| = S_0 + 1$, we have

$$H(X_S | Y_{S^c}) \geq \frac{1}{S_0} \sum_{\lambda \in S} \left(\sum_{k \in \mathcal{L}_\lambda} H(V_{\{k\},:} | V_{\{k\}, \mathcal{M}_\lambda \cup \mathcal{M}_{S^c}}) + H(X_{S\setminus\{\lambda\}} | Y_{(S\setminus\{\lambda\})^c}) \right). \tag{D.2}$$

For each $\lambda \in S$, the first term inside the round brackets in (D.2) can be lower bounded as

$$\sum_{k \in \mathcal{L}_\lambda} H(V_{\{k\},:} | V_{\{k\}, \mathcal{M}_\lambda \cup \mathcal{M}_{S^c}}) \overset{(a)}{=} L_\lambda T \sum_{j=0}^{S_0} a_{S\setminus\{\lambda\}}^{(j)} \tag{D.3}$$

$$\geq L_\lambda T \sum_{j=1}^{S_0} a_{S\setminus\{\lambda\}}^{(j)}, \tag{D.4}$$

where (a) is due to the independence of the intermediate values.

Instead, the second term inside the round brackets in (D.2) can be lower bounded by the assumption that the lemma is valid for all subsets of size S_0:

$$H(X_{S\setminus\{\lambda\}} | Y_{(S\setminus\{\lambda\})^c}) \geq T \sum_{q \in S\setminus\{\lambda\}} \sum_{j=1}^{S_0} a_{S\setminus\{\lambda\}}^{(j,q)} \frac{\sum_{p \in S\setminus\{\lambda\}} L_p - j \cdot L_q}{j^2}. \tag{D.5}$$

Thus, employing equations (D.4) and (D.5) in (D.2), we obtain

$$H(X_S|Y_{S^c}) \geq \frac{1}{S_0}\sum_{\lambda\in S}\left(\sum_{k\in\mathcal{L}_\lambda} H(V_{\{k\},:}|V_{\{k\},\mathcal{M}_\lambda\cup\mathcal{M}_{S^c}}) + H(X_{S\setminus\{\lambda\}}|Y_{(S\setminus\{\lambda\})^c})\right) \tag{D.6}$$

$$\geq \frac{T}{S_0}\sum_{\lambda\in S}\left(L_\lambda\sum_{j=1}^{S_0} a^{(j)}_{S\setminus\{\lambda\}} + \sum_{q\in S\setminus\{\lambda\}}\sum_{j=1}^{S_0} a^{(j,q)}_{S\setminus\{\lambda\}}\frac{\sum_{p\in S\setminus\{\lambda\}} L_p - j\cdot L_q}{j^2}\right) \tag{D.7}$$

$$\overset{(a)}{=} \frac{T}{S_0}\sum_{j=1}^{S_0}\frac{1}{j}\left(\sum_{\lambda\in S}\sum_{q\in S\setminus\{\lambda\}}\frac{a^{(j,q)}_{S\setminus\{\lambda\}}}{j}\left(\sum_{p\in S\setminus\{\lambda\}} L_p + jL_\lambda - jL_q\right)\right) \tag{D.8}$$

$$= \frac{T}{S_0}\sum_{j=1}^{S_0}\frac{1}{j}\left(\sum_{\lambda\in S}\sum_{q\in S\setminus\{\lambda\}}\frac{a^{(j,q)}_{S\setminus\{\lambda\}}}{j}\left(\sum_{p\in S} L_p + (j-1)L_\lambda - jL_q\right)\right) \tag{D.9}$$

where (a) is due to the equality $a^{(j)}_{S\setminus\{\lambda\}} = \frac{1}{j}\sum_{q\in S\setminus\{\lambda\}} a^{(j,q)}_{S\setminus\{\lambda\}}$.

Let us focus on the term

$$\sum_{\lambda\in S}\sum_{q\in S\setminus\{\lambda\}} a^{(j,q)}_{S\setminus\{\lambda\}} L_q.$$

The following equalities hold.

$$\sum_{\lambda\in S}\sum_{q\in S\setminus\{\lambda\}} a^{(j,q)}_{S\setminus\{\lambda\}} L_q \tag{D.10}$$

$$= j\sum_{\lambda\in S}\sum_{q\in S\setminus\{\lambda\}}\frac{1}{j}L_q\cdot\sum_{n=1}^{N}\mathbb{1}(\text{file } n \text{ is only mapped by } j \text{ nodes in } S\setminus\{\lambda\} \text{ including node } q) \tag{D.11}$$

$$= \sum_{n=1}^{N}\sum_{\lambda\in S}\sum_{q\in S\setminus\{\lambda\}} L_q\mathbb{1}(\text{file } n \text{ is only mapped by } j \text{ nodes in } S\setminus\{\lambda\} \text{ including node } q) \tag{D.12}$$

$$= \sum_{n=1}^{N}\sum_{\lambda\in S}\sum_{q\in S\setminus\{\lambda\}} L_q\mathbb{1}(\text{file } n \text{ is only mapped by } j \text{ nodes in } S \text{ including node } q) \times \mathbb{1}(\text{file } n \text{ is not mapped by node } \lambda) \tag{D.13}$$

$$= \sum_{n=1}^{N}\sum_{\lambda\in S}\sum_{q\in S} L_q\mathbb{1}(\text{file } n \text{ is only mapped by } j \text{ nodes in } S \text{ including node } q) \times \mathbb{1}(\text{file } n \text{ is not mapped by node } \lambda) \tag{D.14}$$

$$= \sum_{q\in S} L_q\sum_{n=1}^{N}\mathbb{1}(\text{file } n \text{ is only mapped by } j \text{ nodes in } S \text{ including node } q)$$

$$\times \sum_{\lambda \in S} \mathbb{1}(\text{file } n \text{ is not mapped by node } \lambda) \tag{D.15}$$

$$= \sum_{q \in S} L_q \sum_{n=1}^{N} \mathbb{1}(\text{file } n \text{ is only mapped by } j \text{ nodes in } S \text{ including node } q)(|S| - j) \tag{D.16}$$

$$= \sum_{q \in S} L_q a_S^{(j,q)} (S_0 + 1 - j). \tag{D.17}$$

Next, we focus on the term

$$\sum_{\lambda \in S} \sum_{q \in S \setminus \{\lambda\}} \frac{a_{S\setminus\{\lambda\}}^{(j,q)}}{j} (j-1) L_\lambda \tag{D.18}$$

$$= \frac{j-1}{j} \sum_{\lambda \in S} L_\lambda \sum_{q \in S \setminus \{\lambda\}} a_{S\setminus\{\lambda\}}^{(j,q)} \tag{D.19}$$

To this end, we define $\mathcal{A}_S^{(j,q)}$ as the set of file indices that are mapped by j nodes in S, including node q. We have that $|\mathcal{A}_S^{(j,q)}| = a_S^{(j,q)}$. Furthermore, it holds that

$$\mathcal{A}_{S\setminus\{\lambda\}}^{(j,q)} = \mathcal{A}_S^{(j,q)} \setminus (\mathcal{A}_S^{(j,q)} \cap \mathcal{A}_S^{(j,\lambda)}),$$

which implies that

$$a_{S\setminus\{\lambda\}}^{(j,q)} = |\mathcal{A}_{S\setminus\{\lambda\}}^{(j,q)}| \tag{D.20}$$

$$= |\mathcal{A}_S^{(j,q)} \setminus (\mathcal{A}_S^{(j,q)} \cap \mathcal{A}_S^{(j,\lambda)})| \tag{D.21}$$

$$= |\mathcal{A}_S^{(j,q)}| - |\mathcal{A}_S^{(j,q)} \cap \mathcal{A}_S^{(j,\lambda)}| \tag{D.22}$$

$$= a_S^{(j,q)} - (j-1)|\mathcal{A}_S^{(j,\lambda)}| \tag{D.23}$$

$$= a_S^{(j,q)} - (j-1) a_S^{(j,\lambda)}. \tag{D.24}$$

Placing the RHS of equation (D.24) into (D.19) we obtain

$$\frac{j-1}{j} \sum_{\lambda \in S} L_\lambda \sum_{q \in S \setminus \{\lambda\}} a_{S\setminus\{\lambda\}}^{(j,q)} \tag{D.25}$$

$$= \frac{j-1}{j} \sum_{\lambda \in S} L_\lambda \sum_{q \in S \setminus \{\lambda\}} \left(a_S^{(j,q)} - (j-1) a_S^{(j,\lambda)} \right) \tag{D.26}$$

$$= \frac{j-1}{j} \sum_{\lambda \in S} L_\lambda \left(\sum_{q \in S} a_S^{(j,q)} - j a_S^{(j,\lambda)} \right) \tag{D.27}$$

$$=\frac{j-1}{j}\left(\sum_{\lambda\in S}L_\lambda\sum_{q\in S}a_S^{(j,q)}-j\sum_{\lambda\in S}L_\lambda a_S^{(j,\lambda)}\right) \tag{D.28}$$

Finally, using (D.17) and (D.28) in (D.9) yields

$$H(X_S|Y_{S^c})\geq\frac{T}{S_0}\sum_{j=1}^{S_0}\frac{1}{j}\left(\sum_{\lambda\in S}\sum_{q\in S\setminus\{\lambda\}}\frac{a_{S\setminus\{\lambda\}}^{(j,q)}}{j}\left(\sum_{p\in S}L_p+(j-1)L_\lambda-jL_q\right)\right) \tag{D.29}$$

$$\begin{aligned}&=\frac{T}{S_0}\sum_{j=1}^{S_0}\frac{1}{j}\Bigg((j-1)\sum_{\lambda\in S}L_\lambda\sum_{q\in S}\frac{a_S^{(j,q)}}{j}-(j-1)\sum_{\lambda\in S}L_\lambda a_S^{(j,\lambda)}+\\&\qquad+\sum_{q\in S}\frac{a_S^{(j,q)}}{j}\sum_{\lambda\in S}L_\lambda(S_0+1-j)-\sum_{q\in S}L_q a_S^{(j,q)}(S_0+1-j)\Bigg)\end{aligned} \tag{D.30}$$

$$=\frac{T}{S_0}\sum_{j=1}^{S_0}\frac{1}{j}\left(S_0\sum_{\lambda\in S}L_\lambda\sum_{q\in S}\frac{a_S^{(j,q)}}{j}-S_0\sum_{q\in S}L_q a_S^{(j,q)}\right) \tag{D.31}$$

$$=T\sum_{j=1}^{S_0}\frac{1}{j}\left(\sum_{\lambda\in S}L_\lambda\sum_{q\in S}\frac{a_S^{(j,q)}}{j}-\sum_{q\in S}L_q a_S^{(j,q)}\right) \tag{D.32}$$

$$=T\sum_{j=1}^{S_0}\sum_{q\in S}a_S^{(j,q)}\frac{\sum_{\lambda\in S}L_\lambda-jL_q}{j^2} \tag{D.33}$$

$$\overset{(a)}{=}T\sum_{j=1}^{S_0+1}\sum_{q\in S}a_S^{(j,q)}\frac{\sum_{\lambda\in S}L_\lambda-jL_q}{j^2} \tag{D.34}$$

where (a) follows from the following equality

$$T\sum_{q\in S}a_S^{(S_0+1,q)}\frac{\sum_{\lambda\in S}L_\lambda-(S_0+1)L_q}{(S_0+1)^2}=\frac{T}{S_0+1}\left(a_S^{(S_0+1)}\sum_{\lambda\in S}L_\lambda-\sum_{q\in S}a_S^{(S_0+1,q)}L_q\right) \tag{D.35}$$

$$\overset{(b)}{=}\frac{T}{S_0+1}\left(a_S^{(S_0+1)}\sum_{\lambda\in S}L_\lambda-a_S^{(S_0+1)}\sum_{q\in S}L_q\right) \tag{D.36}$$

$$=0, \tag{D.37}$$

and where in (b) we have used the fact that $a_S^{(|S|)}=a_S^{(|S|,q)}$ for any $q\in S$. The equation in (D.34) proves that if the lemma holds for any subset $S\subseteq[\Lambda]$ of size $|S|=S_0$, then it also holds for a subset $S\subseteq[\Lambda]$ of size $|S|=S_0+1$. This fact, combined with the fact that the lemma holds for any subset $S=\{\lambda\},\lambda\in[\Lambda]$, completes the proof by induction.

Bibliography

[1] Cisco, "Cisco visual networking index: Forecast and trends, 2017-2022 white paper," 2019. [Online]. Available: https://www.cisco.com

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

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

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

[5] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "The exact rate-memory tradeoff for caching with uncoded prefetching," *IEEE Transactions on Information Theory*, vol. 64, no. 2, pp. 1281–1296, Feb 2018.

[6] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, Aug 2015.

[7] S. P. Shariatpanahi, S. A. Motahari, and B. H. Khalaj, "Multi-server coded caching," *IEEE Transactions on Information Theory*, vol. 62, pp. 7253–7271, Dec 2016.

[8] S. P. Shariatpanahi, G. Caire, and B. H. Khalaj, "Physical-layer schemes for wireless coded caching," *IEEE Transactions on Information Theory*, pp. 1–1, 2018.

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

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

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

[12] V. Ravindrakumar, P. Panda, N. Karamchandani, and V. Prabhakaran, "Fundamental limits of secretive coded caching," in *IEEE International Symposium on Information Theory, (ISIT)*, 2016, pp. 425–429.

[13] V. Ravindrakumar, P. Panda, N. Karamchandani, and V. M. Prabhakaran, "Private coded caching," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 685–694, 2018.

[14] Q. Yan and D. Tuninetti, "Fundamental limits of caching for demand privacy against colluding users," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 1, pp. 192–207, 2021.

[15] M. Ji, G. Caire, and A. F. Molisch, "Fundamental limits of caching in wireless D2D networks," *IEEE Transactions on Information Theory*, vol. 62, no. 2, pp. 849–869, Feb 2016.

[16] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, Jan 2018.

[17] P. Krishnan, V. Lalitha, and L. Natarajan, "Coded data rebalancing: Fundamental limits and constructions," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 640–645.

[18] K. Wan, D. Tuninetti, M. Ji, and P. Piantanida, "Fundamental limits of distributed data shuffling," 2018. [Online]. Available: https://arxiv.org/abs/1807.00056

[19] R. Sun, H. Zheng, j. Liu, X. Du, and M. Guizani, "Placement delivery array for the coded caching scheme in medical data sharing," in *Neural Computing and Application*, 2020, pp. 867–878.

[20] [Online]. Available: https://cadami.net/

[21] N. Naderializadeh, M. A. Maddah-Ali, and A. S. Avestimehr, "Fundamental Limits of Cache-Aided Interference Management," *IEEE Transactions on Information Theory*, vol. 63, no. 5, pp. 3092–3107, 2017.

[22] E. Lampiris, A. Bazco-Nogueras, and P. Elia, "Resolving the Feedback Bottleneck of Multi-Antenna Coded Caching," *arXiv preprint arXiv:1811.03935*, 2018. [Online]. Available: http://arxiv.org/abs/1811.03935

[23] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, 2008.

[24] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[25] N. Golrezaei, K. Shanmugam, A. Dimakis, A. Molisch, and G. Caire, "Femtocaching: Wireless video content delivery through distributed caching helpers," in *IEEE Conference on Computer Communications, (INFOCOM)*, March 2012, pp. 1107–1115.

[26] J. Hachem, N. Karamchandani, and S. Diggavi, "Coded caching for multi-level popularity and access," *IEEE Transactions on Information Theory*, vol. 63, pp. 3108–3141, May 2017.

[27] H. Xu, C. Gong, and X. Wang, "Efficient file delivery for coded prefetching in shared cache networks with multiple requests per user," 2018. [Online]. Available: http://arxiv.org/abs/1803.09408

[28] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, "Caching-aided coded multicasting with multiple random requests," in *IEEE Information Theory Workshop, (ITW)*, May 2015, pp. 1–5.

[29] A. Sengupta and R. Tandon, "Improved approximation of storage-rate tradeoff for caching with multiple demands," *IEEE Transactions on Communications*, vol. 65, no. 5, pp. 1940–1955, May 2017.

[30] Y. Wei and S. Ulukus, "Coded caching with multiple file requests," in *55th Annual Allerton Conference on Communication, Control, and Computing*, Oct 2017, pp. 437–442.

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

[32] 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 Transactions on Communications*, vol. 67, no. 8, pp. 5297–5310, 2019.

[33] E. Lampiris and P. Elia, "Adding transmitters dramatically boosts coded-caching gains for finite file sizes," *IEEE Journal on Selected Areas in Communications (Special Issue on Caching)*, vol. 36, no. 6, pp. 1176–1188, June 2018.

[34] X. Peng, J. Zhang, S. H. Song, and K. B. Letaief, "Cache size allocation in backhaul limited wireless networks," in *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1–6.

[35] T. Liu, S. Zhou, and Z. Niu, "Joint optimization of cache allocation and content placement in urban vehicular networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6.

[36] J. Liao, K. Wong, Y. Zhang, Z. Zheng, and K. Yang, "Coding, multicast, and cooperation for cache- enabled heterogeneous small cell networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 10, pp. 6838–6853, 2017.

[37] S. Saeedi Bidokhti, M. Wigger, and A. Yener, "Benefits of cache assignment on degraded broadcast channels," *IEEE Transactions on Information Theory*, vol. 65, no. 11, pp. 6999–7019, 2019.

[38] A. Tang, S. Roy, and X. Wang, "Coded caching for wireless backhaul networks with unequal link rates," *IEEE Transactions on Communications*, vol. 66, no. 1, pp. 1–13, 2018.

[39] A. M. Ibrahim, A. A. Zewail, and A. Yener, "Centralized coded caching with heterogeneous cache sizes," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, 2017, pp. 1–6.

[40] L. Zheng, Q. Chen, Q. Yan, and X. Tang, "Decentralized coded caching scheme with heterogeneous file sizes," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 1, pp. 818–827, 2020.

[41] E. Parrinello, A. Ünsal, and P. Elia, "Optimal coded caching in heterogeneous networks with uncoded prefetching," in *IEEE Information Theory Workshop, (ITW)*, 2018.

[42] E. Parrinello, A. Ünsal, and P. Elia, "Fundamental limits of coded caching with multiple antennas, shared caches and uncoded prefetching," *IEEE Transactions on Information Theory*, vol. 66, no. 4, pp. 2252–2268, 2020.

[43] E. Parrinello and P. Elia, "Coded caching with optimized shared-cache sizes," in *2019 IEEE Information Theory Workshop (ITW)*, 2019, pp. 1–5.

[44] E. Parrinello, A. Bazco-Nogueras, and P. Elia, "Fundamental limits of topology-aware shared-cache networks," *to be submitted to IEEE Transactions on Information Theory*, 2021.

[45] B. Serbetci, E. Parrinello, and P. Elia, "Multi-access coded caching: gains beyond cache-redundancy," in *2019 IEEE Information Theory Workshop (ITW)*, 2019, pp. 1–5.

[46] E. Parrinello, P. Elia, and E. Lampiris, "Extending the optimality range of multi-antenna coded caching with shared caches," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 1675–1680.

[47] A. Tolli, S. P. Shariatpanahi, J. Kaleva, and B. H. Khalaj, "Multi-antenna interference management for coded caching," *IEEE Transactions on Wireless Communications*, vol. 19, no. 3, pp. 2091–2106, 2020.

[48] E. Lampiris, P. Elia, and G. Caire, "Bridging the gap between multiplexing and diversity in finite SNR multiple antenna coded caching," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2019, pp. 1272–1277.

[49] M. Salehi, E. Parrinello, S. P. Shariatpanahi, P. Elia, and A. Tölli, "Low-complexity high-performance cyclic caching for large miso systems," 2020.

[50] E. Parrinello, A. Ünsal, and P. Elia, "Optimal coded caching under statistical QoS information," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 2987–2991.

[51] E. Parrinello and P. Elia, "New optimality results for heterogeneous coded distributed computing," *Manuscript in preparation*, 2021.

[52] K. Wan, D. Tuninetti, and P. Piantanida, "On the optimality of uncoded cache placement," in *IEEE Information Theory Workshop, (ITW)*, 2016, pp. 161–165.

[53] S. Jin, Y. Cui, H. Liu, and G. Caire, "Order-optimal decentralized coded caching schemes with good performance in finite file size regime," in *IEEE Global Communications Conference, (GLOBECOM)*, Dec 2016, pp. 1–7.

[54] M. Li, L. Ong, and S. J. Johnson, "Cooperative multi-sender index coding," 2018. [Online]. Available: https://arxiv.org/abs/1701.03877v4

[55] P. Sadeghi, F. Arbabjolfaei, and Y. H. Kim, "Distributed index coding," in *IEEE Information Theory Workshop, (ITW)*, Sept 2016, pp. 330–334.

[56] N. S. Karat, S. Dey, A. Thomas, and B. S. Rajan, "An optimal linear error correcting delivery scheme for coded caching with shared caches," *CoRR*, vol. abs/1901.03188, 2019. [Online]. Available: http://arxiv.org/abs/1901.03188

[57] B. Asadi and L. Ong, "Centralized caching with shared caches in heterogeneous cellular networks," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2019, pp. 1–5.

[58] A. M. Ibrahim, A. A. Zewail, and A. Yener, "Coded placement for systems with shared caches," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[59] K. Wan, D. Tuninetti, M. Ji, and G. Caire, "On the fundamental limits of fog-ran cache-aided networks with downlink and sidelink communications," *IEEE Transactions on Information Theory*, vol. 67, no. 4, pp. 2353–2378, 2021.

[60] M. Dutta and A. Thomas, "Decentralized coded caching for shared caches," *IEEE Communications Letters*, vol. 25, no. 5, pp. 1458–1462, 2021.

[61] A. Malik, B. Serbetci, E. Parrinello, and P. Elia, "Fundamental limits of stochastic shared-cache networks," *IEEE Transactions on Communications*, pp. 1–1, 2021.

[62] J. Zhang and P. Elia, "Fundamental limits of cache-aided wireless BC: Interplay of coded-caching and CSIT feedback," *IEEE Transactions on Information Theory*, vol. 63, no. 5, pp. 3142–3160, May 2017.

[63] C. Niculescu, "A new look at newton's inequalities," *Journal of Inequalities in Pure & Applied Mathematics (JIPAM)*, vol. 1, 01 2000.

[64] K. Wan, D. Tuninetti, and P. Piantanida, "On caching with more users than files," in *2016 IEEE International Symposium on Information Theory (ISIT)*, 2016, pp. 135–139.

[65] H. Joudeh, E. Lampiris, P. Elia, and G. Caire, "Fundamental limits of wireless caching under mixed cacheable and uncacheable traffic," *IEEE Transactions on Information Theory*, pp. 1–1, 2021.

[66] K. S. Reddy and N. Karamchandani, "On the exact rate-memory trade-off for multi-access coded caching with uncoded placement," in *2018 International Conference on Signal Processing and Communications (SPCOM)*, July 2018.

[67] S. Sasi and B. S. Rajan, "An improved multi-access coded caching with uncoded placement," *arXiv preprint arXiv:2009.05377*, 2020.

[68] M. Cheng, D. Liang, K. Wan, M. Zhang, and G. Caire, "A novel transformation approach of shared-link coded caching schemes for multiaccess networks," *arXiv preprint arXiv:2012.04483*, 2020.

[69] K. S. Reddy and N. Karamchandani, "Structured index coding problems and multi-access coded caching," in *2020 IEEE Information Theory Workshop (ITW)*, 2021, pp. 1–5.

[70] S. Sasi and B. S. Rajan, "Multi-access coded caching scheme with linear sub-packetization using pdas," 2021.

[71] D. Katyal, P. N. Muralidhar, and B. S. Rajan, "Multi-access coded caching schemes from cross resolvable designs," *IEEE Transactions on Communications*, vol. 69, no. 5, pp. 2997–3010, 2021.

[72] K. Namboodiri and B. S. Rajan, "Multi-access coded caching with secure delivery," *arXiv preprint arXiv:2105.05611*, 2021.

[73] ——, "Multi-access coded caching with demand privacy," *arXiv preprint arXiv:2107.00226*, 2021.

[74] D. Liang, K. Wan, M. Cheng, and G. Caire, "Multiaccess coded caching with private demands," *arXiv preprint arXiv:2105.06282*, 2021.

[75] M. Salehi, A. Tölli, and S. P. Shariatpanahi, "A multi-antenna coded caching scheme with linear subpacketization," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[76] S. Mohajer and I. Bergel, "MISO Cache-Aided Communication with Reduced Subpacketization," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[77] M. Salehi, A. Tolli, S. P. Shariatpanahi, and J. Kaleva, "Subpacketization-rate trade-off in multi-antenna coded caching," in *2019 IEEE Global Communications Conference, GLOBECOM 2019 - Proceedings*. IEEE, 2019, pp. 1–6.

[78] M. J. Salehi, A. Tolli, and S. P. Shariatpanahi, "Subpacketization-beamformer interaction in multi-antenna coded caching," in *2nd 6G Wireless Summit 2020: Gain Edge for the 6G Era, 6G SUMMIT 2020*, 2020, pp. 1–5.

[79] S. A. Vorobyov, A. B. Gershman, and Z.-Q. Luo, "Robust adaptive beamforming using worst-case performance optimization: A solution to the signal mismatch problem," *IEEE transactions on signal processing*, vol. 51, no. 2, pp. 313–324, 2003.

[80] P. Komulainen, "Coordinated multi-antenna techniques for cellular networks : Pilot signaling and decentralized optimization in TDD mode," Ph.D. dissertation, University of Oulu, 2013. [Online]. Available: http://urn.fi/urn:isbn:9789526202815

[81] M. Bengtsson and B. Ottersten, "Optimal and Suboptimal Transmit Beamforming," no. July, pp. 18–1, 2001.

[82] A. Wiesel, Y. C. Eldar, and S. Shamai, "Linear precoding via conic optimization for fixed MIMO receivers," *IEEE Transactions on Signal Processing*, vol. 54, no. 1, pp. 161–176, 2006.

[83] R. D. Yates, "A framework for uplink power control in cellular radio systems," *IEEE Journal on selected areas in communications*, vol. 13, no. 7, pp. 1341–1347, 1995.

[84] Q. Yang and D. Gündüz, "Coded caching and content delivery with heterogeneous distortion requirements," *IEEE Transactions on Information Theory*, vol. 64, no. 6, pp. 4347–4364, 2018.

[85] M. M. Amiri and D. Gündùz, "On the capacity region of a cache-aided gaussian broadcast channel with multi-layer messages," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1909–1913.

[86] M. Bayat, C. Yapar, and G. Caire, "Spatially scalable lossy coded caching," in *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, Aug 2018, pp. 1–6.

[87] A. M. Ibrahim, A. A. Zewail, and A. Yener, "On coded caching with heterogeneous distortion requirements," in *2018 Information Theory and Applications Workshop (ITA)*, Feb 2018, pp. 1–9.

[88] D. Cao, D. Zhang, P. Chen, N. Liu, W. Kang, and D. Gunduz, "Coded caching with heterogeneous cache sizes and link qualities: The two-user case," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1545–1549.

[89] F. Arbabjolfaei, B. Bandemer, Y. H. Kim, E. Şaşoğlu, and L. Wang, "On the capacity region for index coding," in *IEEE International Symposium on Information Theory, (ISIT)*, Jul 2013, pp. 962–966.

[90] M. Kiamari, C. Wang, and A. S. Avestimehr, "On heterogeneous coded distributed computing," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–7.

[91] F. Xu and M. Tao, "Heterogeneous coded distributed computing: Joint design of file allocation and function assignment," 2019. [Online]. Available: https://arxiv.org/abs/1908.06715

[92] N. Woolsey, R.-R. Chen, and M. Ji, "Coded distributed computing with heterogeneous function assignments," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[93] F. Xu, S. Shao, and M. Tao, "New results on the computation-communication tradeoff for heterogeneous coded distributed computing," *IEEE Transactions on Communications*, vol. 69, no. 4, pp. 2254–2270, 2021.

[94] J. Xu, L. Fu, and X. Wang, "Distributed computing with heterogeneous servers," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[95] N. Woolsey, R.-R. Chenb, M. Jic, N. Woolsey, R.-R. Chenb, and M. Jic, "A combinatorial design for cascaded coded distributed computing on general networks," *IEEE Transactions on Communications*, pp. 1–1, 2021.

[96] 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 Transactions on Communications*, vol. 67, no. 12, pp. 8179–8192, 2019.

[97] I. Newton, G. Baermann, R. Boškovic, G. Campbell, J. de Castillon, J. Colson, E. Halley, A. Kästner, C. MacLaurin, A. de Moivre *et al.*, *Arithmetica universalis sive de compositione et resolutione arithmetica liber*, ser. Arithmetica universalis, sive De compositione et resolutione arithmetica. apud Marcum Michaelem Rey, 1761.

[98] R. Stanley, "Log-concave and unimodal sequences in algebra, combinatorics, and geometry," *Annals of the New York Academy of Sciences*, vol. 576, pp. 500 – 535, 12 2006.

[99] M. Lin and N. S. Trudinger, "On some inequalities for elementary symmetric functions," *Bulletin of the Australian Mathematical Society*, vol. 50, no. 2, pp. 317–326, 1994.