University of Thessaly

Doctoral Thesis

Intelligent information caching in novel network architectures

Author:
Konstantinos Poularakis

Supervisor:
Dr. Athanasios Korakis

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Department of Electrical and Computer Engineering

January 2016

Declaration of Authorship

I, Konstantinos Poularakis, declare that this thesis titled, 'Intelligent information caching in novel network architectures' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UNIVERSITY OF THESSALY

Abstract

Department of Electrical and Computer Engineering

Doctor of Philosophy

Intelligent information caching in novel network architectures

by Konstantinos Poularakis

A significant portion of today's network traffic is due to recurring downloads of a few popular contents (e.g., movies, video clips and daily news). It has been observed that replicating the latter in caches installed at network edge -close to users- can drastically reduce network bandwidth usage and improve content access delay. In this thesis, such caching architectures are studied starting with the hierarchical structure architecture and moving to emerging architectures that enable caching at the wireless edge. In particular, we develop mechanisms that make caching decisions about *where* and *which* content item to cache and manage the routing of content among caches. The novelty of our work lies on exploiting the special characteristics of user content access such as (i) the diversity of user demand in terms of the required quality level of content (e.g., spatial resolution and frame rate of video), (ii) the concurrency in accessing content across users which allows serving multiple requesters via a common multicast stream and (iii) the regularity of user mobility patterns which can be used for predicting future access to cache-endowed nodes. These are cutting-edge approaches that can achieve significant performance and cost-reduction benefits over the state-of-the-art methods.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Περίληψη

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Διδακτορικό Δίπλωμα

Ευφυής αποθήκευση πληροφορίας σε νέες αρχιτεκτονικές δικτύων

Κωνσταντίνος Πουλαράκης

Ένα σημαντικό μέρος του διαδικτυακού φόρτου σήμερα οφείλεται σε επαναλαμβανόμενες λήψεις ενός μικρού αριθμού δημοφιλών αρχείων (π.χ. ταινίες, βίντεο κλιπ και νέα της ημέρας). Έχει παρατηρηθεί ότι η τοποθέτηση αντιγράφων των τελευταίων σε κρυφές μνήμες (caches) εγκατεστημένες στα άκρα του δικτύου - κοντά στους χρήστες - μπορεί να μειωσεί δραστικά την κατανάλωση εύρους ζώνης του δικτύου και να βελτιώσει την καθυστέρηση πρόσβασης των χρηστών. Στην παρούσα διατριβή, μελετώνται αρχιτεκτονικές προσωρινής αποθήκευσης περιεχομένου (caching) ξεκινώντας από τα δίκτυα ιεραρχικής δομής και συνεχίζοντας σε καινοτόμες αρχιτεκτονικές οι οποίες επιτρέπουν την προσωρινή αποθήκευση στα άκρα των ασύρματων δικτύων. Συγκεκριμένα, αναπτύσσουμε μηχανισμούς οι οποίοι λαμβάνουν τις αποφάσεις σχετικά με το *που* και *ποιο* αρχείο θα αποθηκευτεί και τη διαχείριση της δρομολόγησης του περιεχομένου στο δίκτυο. Η καινοτομία έγκειται στην αξιοποίηση των ιδιαίτερων χαρακτηριστικών της πρόσβασης περιεχομένου από τους χρήστες, όπως (i) η ποικιλομορφία των απαιτήσεων των χρηστών όσον αφορά το επίπεδο ποιότητας του περιεχομένου (π.χ., χωρική ανάλυση και ρυθμός ανανέωσης πλαισίων βίντεο), (ii) οι ταυτόχρονες αιτήσεις περιεχομένου μεταξύ των χρηστών γεγονός το οποίο επιτρέπει την εξυπηρέτηση πολλαπλών αιτούντων μέσω μιας κοινής αποστολής (multicast) και (iii) τα πρότυπα κινητικότας των ασύρματων χρηστών τα οποία μπορούν να χρησιμοποιηθούν για την πρόβλεψη της μελλοντικής πρόσβασης στις caches. Αυτές είναι προσεγγίσεις αιχμής οι οποίες δύναται να επιτύχουν σημαντική βελτίωση στην απόδοση και μείωση του κόστους του δικτύου σε σύγκριση με τις υπάρχουσες τεχνικές.

Acknowledgements

I would like to thank all the members of the thesis committee; Prof. Athanasios Korakis, Iordanis Koutsopoulos, Leonidas Georgiadis, Elias Houstis, Aikaterini Housti, Spyros Lalis and Antonios Argyriou.

I am deeply indebted to Prof. Leandros Tassiulas for first taking me as an undergraduate and Master's student, guiding me through my first steps in research and sharing his numerous ideas, many of which form the foundations of this thesis.

Last, I would like to thank the "Alexander S. Onassis Public Benefit Foundation" for the financial support through a scholarship for the years 2013-2015.

Publications

The results, the ideas and figures are included in the following publications:

International Journals

[J.1] K. Poularakis, L. Tassiulas, "On the Complexity of Optimal Content Placement in Hierarchical Caching Networks", *IEEE Transactions on Communications, 2016, subject to minor revision.*

[J.2] K. Poularakis, G. Iosifidis, I. Pefkianakis, L. Tassiulas, Martin May, "Mobile Data Offloading through Caching in Residential 802.11 Wireless Networks", *IEEE Transactions on Network and Service Management, 2016, to appear.*

[J.3] K. Poularakis, G. Iosifidis, V. Sourlas, L. Tassiulas, "Exploiting Caching and Multicast for 5G Wireless Networks", *IEEE Transactions on Wireless Communications, 2016, to appear.*

[J.4] K. Poularakis, L. Tassiulas, "Code, Cache and Deliver on the Move: A Novel Caching Paradigm in Hyper-Dense Small-cell Networks", *IEEE Transactions on Mobile Computing, 2016, subject to minor revision.*

[J.5] K. Poularakis, L. Tassiulas, "Cooperation and information replication in wireless networks", *Royal Society, Philosophical Transactions A, 2016, to appear.*

[J.6] K. Poularakis, G. Iosifidis, L. Tassiulas, "Approximation Algorithms for Mobile Data Caching in Small Cell Networks", *IEEE Transactions on Communications, vol. 62, no. 10, pp. 3665-3677, October 2014.*

International Conferences

[C.1] K. Poularakis, G. Iosifidis, A. Argyriou, I. Koutsopoulos, L. Tassiulas, "Caching and Operator Cooperation Policies for Layered Video Content Delivery", *IEEE International Conference on Computer Communications (INFOCOM), 2016, to appear (***Acceptance rate: 18.2%***).*

[C.2] K. Poularakis, G. Iosifidis, A. Argyriou, L. Tassiulas, "Video Delivery over Heterogeneous Cellular Networks: Optimizing Cost and Performance", *IEEE International Conference on Computer Communications (INFOCOM), pp. 1078-1086, 2014 (***Acceptance rate: 19.3%***).*

[C.3] K. Poularakis, G. Iosifidis, V. Sourlas, L. Tassiulas, "Multicast-aware Caching for Small Cell Networks", *IEEE Wireless Communications and Networking Conference (WCNC), pp. 2300-2305, 2014.*

[C.4] K. Poularakis, G. Iosifidis, L. Tassiulas, "Approximation Caching and Routing Algorithms for Massive Mobile Data Delivery", *IEEE Global Communications Conference (GLOBECOM), pp. 3534-3539, 2013.*

[C.5] K. Poularakis, L. Tassiulas, "Exploiting User Mobility for Wireless Content Delivery", *IEEE International Symposium on Information Theory (ISIT), pp. 1017-1021, 2013.*

Contents

To my family, Niko, Christina, Stergio and Olga, and to Mihali.

Chapter 1

Introduction

Contents

1.1 Motivation

Today, we are witnessing an explosive growth in global internet traffic, that is expected to nearly triple between 2014 to 2019 [1]. This trend is mainly fueled by the penetration of fixed-line and mobile broadband packages and the popularity of modern communication devices with large screens and rich multimedia capabilities (smart phones, tablets, notebooks, in-vehicle communication systems). These developments herald the advent of a new era in communication systems, with novel challenges for content delivery over wired and wireless networks. Namely, network operators (NOs) are facing a *bandwidth crunch* due to the gap between network capacity of their networks and demand. This results in degradation of user service quality and threatens the economic viability of NOs.

Traditional methods for increasing network capacity, such as technology upgrades (e.g., from DSL to Fiber, from WCDMA to LTE, etc.) and additional spectrum acquisition, may be prohibitively expensive, require significant time to implement, or even be infeasible due, for example, to prior spectrum allocation for other uses (television, radio, military purposes). More importantly these methods will most probably be outpaced by

the continuing traffic increase, requiring novel *out-of-the-box* ideas and business models for moving the field forward.

An alternative approach for mitigating the effects of traffic growth is to replicate *popular* content items in *caches* installed at network edge -close to users. This way, the cached content can be rapidly delivered to requesters, decreasing the bandwidth consumption in the core network. Clearly, such caching schemes are more effective when a few popular contents (e.g., movies, video clips, daily news, etc.) attract an important portion of network traffic. The idea of in-network caching is traced back to the early 90's, when proxy servers were used as caches to improve the scalability of world wide web. Since then, various caching architectures have been proposed, differing in the level of cooperation between the caches in serving user requests [2]. More recently, the concept of information-centric networks (ICNs) appeared, which aims to change the way of accessing the content on the internet, by uniquely naming contents and replicating them almost ubiquitously throughout the network [3], [4]. Today, storage is considered as a network resource that is managed similarly to link bandwidth, and related products and technical solutions exist [5]. The large-scale deployment of in-network caching architectures is favoured by the downward slope in storage space price (a few tens of US dollars per terabyte currently [6]).

Caching has been used successfully in Content Distribution Networks (CDNs) to manage the tremendous growth in broadband data consumption, by replicating popular content in various locations and in effect closer to users [7]. At the same time, caching is expected to play a crucial role in emerging 5G wireless communication networks. Indeed, there are currently various proposals for using caching at the network core so as to reduce transit bandwidth costs, at the base stations to overcome the limited backhaul capacity problem [8], and even at the users' devices in order to exploit their devices' capabilities [9]. There is currently growing consensus that such architectural innovations can multiply network throughput, reduce networks' expenditures, and improve user perceived performance.

Interestingly, the wireless industry has already begun to commercialize systems that enable caching in the radio access network (RAN). Notable examples include Altobridge's "Data at-the-Edge" solution [10], Nokia Siemens Networks' liquid application solution [11] and Saguna Networks' Open RAN platform [12] with caching at the LTE base stations. There have been also some initial developments of such technology at the WiFi Access Point (AP) side. For example, the Linksys Smart WiFi [13] and the Hi-WiFi [14] routers can be connected with external storage devices using their USB ports. Importantly, their advanced operating system is capable of running various applications to customize content caching schemes. These efforts were encouraged by measurement

studies that indicated substantial performance benefits, up to two thirds reduction in mobile traffic, by using RAN caching in 3G [15] and 4G [16] networks.

Despite the plethora of work in this field, there are still many questions to be answered. For example, *how to make caching decisions about where and which content item to cache and manage the routing of content among caches?* Second, *how the specific requirements of different types of user applications impact the efficiency of the caching algorithms?* For example, in the context of video streaming applications, different users may ask for different quality-levels (e.g., spatial resolutions, frame rates, etc.) of the same video file and therefore the caching problem needs to be revisited to consider caching the different video-quality segments. Third, *how the caching algorithms developed for wired networks can be extended to the wireless domain?* The latter question is not trivial to answer due to the unique characteristics of the wireless networks, such as (i) the broadcast nature of the wireless medium, which allows serving many concurrent requests via a common *multicast* stream, and (ii) the *mobility* of the users who may be handed-off from one base station to another before data transfer is finished. In the following subsection, we provide the outline of our work for tackling these challenges.

1.2 Outline and Contributions

This thesis consists of two main parts dealing with caching in wired and wireless networks respectively. We first study hierarchical wired networks, where caches may be installed at multiple hierarchy levels (Chapter 2). Then, we move to wireless networks and define the joint problem of caching content at cellular base stations or WiFi access points, and routing content to users (Chapter 3). An extension to the case of diverse requests for videos that are available at multiple qualities is provided in Chapter 4. Finally, caching algorithms that exploit the broadcast/multicast nature of wireless medium and the predictability of user mobility patterns are developed in Chapters 5 and 6 respectively.

Below, we briefly discuss the contribution of our work in each one of the five chapters.

Chapter 2. Hierarchical Caching: Hierarchical topologies have been applied in many systems that provide massive content delivery services, such as IPTV and video on Demand (VoD) [17]. In such systems, requests for content files are generated at the bottom-level nodes of the hierarchy and are routed *upwards* until they reach a cache-node (e.g., an intermediate office (IO), a central office (CO) or a digital subscriber line access multiplexer (DSLAM)) that stores the requested file. If none of the accessed nodes has stored the respective file, a distant content server is triggered to serve the request.

Since the latter option incurs extra processing overhead and raises scalability concerns, our goal in this chapter is to reduce the load of the server by serving as many requests as possible by the caches. The currently best performing caching methods achieve an approximation ratio close to 2, i.e., in the worst case half of the optimal number of requests are served by the caches [18]. In our recent work [19], we showed that this problem is NP-Hard, we uncovered a tractable special case of caches installed on a single hierarchy path and developed an algorithm achieving a provably better approximation ratio than the best-known counterparts.

Chapter 3. Joint Caching and Routing in Wireless Networks: In order to cope with the mobile data traffic explosion mobile network operators deploy small cell base stations (SBSs), such as pico-cells and femto-cells, or WiFi Access Points (APs), which operate in conjunction with the collocated macrocell base stations [20]. Local caching of popular content items at the SBSs has been proposed in order to decrease the costly transmissions from the macrocell base stations without requiring high capacity backhaul links for connecting the SBSs with the core network. However, the caching policy design is a challenging problem especially if one considers realistic parameters such as the bandwidth capacity constraints of the SBSs that can be reached in congested urban areas. In our recent work [21], we studied the joint caching and request routing problem aimed at maximizing the number of requests served by the cache-endowed SBSs. We identified a connection of this problem to an instance of the facility location problem. This paved the road for exploiting the broad literature in facility location algorithms for deriving caching algorithms with provable approximation ratios. We also proposed leasing cache space and wireless bandwidth from residential WiFi APs for offloading mobile data in [22]. To encourage residential users to contribute their cache and bandwidth resources, we designed monetary incentive (reimbursement) schemes. Using a novel WiFi usage dataset collected from 167 residential users, we showed that in densely-populated areas, our proposal can reduce operator's cost by a factor of 2.

Chapter 4. Caching Layered Video: Video delivery to mobile users is one of the largest challenges that network operators face today. In contrast to other types of content, video should be available in various qualities since users often have different quality requirements (spatial resolutions, frame rates, etc.). To achieve this, every video can be encoded into multiple versions which differ in quality and rate (*versions*). Another option is scalable video coding (SVC) (*layers*) where each video is encoded into different layers which, when combined, produce a quality that increases as more layers are used. This technique introduces an encoding overhead but offers network flexibility since the layers of each video can be cached at different base stations and/or routed over different paths. The MNO can use versions, layers or a mixture of them to cache and deliver video to mobile users. In our recent work [23], we showed that the caching problem obtains

an interesting new twist with the advent of SVC, with both the solution space and the objective function of the problem being different. We derived novel approximation algorithms using a connection to a knapsack-type problem and a technique that partitions the amount of cache space of a node dedicated to own and others' content. Going one step further in [24], we revisited the joint caching and routing problem introduced in Chapter 3 for the case of video delivery with the goal of minimizing a balanced objective of average delay and servicing cost. The numerical results indicated that versions and layers may have different impact on the delay and servicing cost, depending on the diversity of users' demand, and that the cost-delay trade off is affected by the network's load.

Chapter 5. Multicast-aware Caching: Many operators take advantage of multicast to efficiently utilize the available bandwidth of their networks in delivering the same content to multiple receivers. Compared to unicast communication, multicast incurs less traffic as the requested file is transmitted to users only once, rather than with many point-to-point transmissions. Intuitively, multicast should be effective when there is significant *concurrency* in accessing information across users; i.e., many users concurrently generate requests for the same content file. Such scenarios are more common during crowded events with a large number of co-located people that are interested in the same contents, e.g., during sporting games, concerts and public demonstrations with often tens of thousand attendees. In our recent work [25], we designed caching algorithms with concerns on multicast transmissions. We showed that the multicast-aware caching problem is NP-Hard and developed solutions with performance guarantees using randomized-rounding techniques. Trace-driven numerical results showed that in presence of massive demand for delay tolerant content, combining caching and multicast can reduce energy consumption of the network. The gains over existing caching schemes are 17.5% when users tolerate delay of three minutes, increasing further with the steepness of content access pattern.

Chapter 6. Mobility-aware Caching: In the emerging hyper-dense cellular network deployments, mobile users may associate to multiple base stations encountered at close times. Clearly, leveraging predictions about the mobility patterns of the users can play a crucial role in further improving the gains that can be acquired from caching in wireless networks. In our recent work [26], we introduced an optimization framework that models user movements via random walks on a Markov chain aimed at minimizing the load of the macro-cell. As the main contribution, we put forward a distributed caching paradigm that leverages user mobility predictions and innovative information-mixing methods based on the principle of network coding. Systematic experiments based on measured traces of human mobility patterns demonstrated that our approach can offload 65% more macro-cell traffic than existing caching schemes in realistic settings.

1.3 Literature Review

Several questions arise when dealing with in-network caching, such as determining in which nodes to install caches, which content files to store in each cache and how often to refresh the cached content. Although all these questions are particularly important for network operators, typically they need to be solved in different timescales. For example, an operator will most probably install caches at some nodes once, but periodically update the cached content, due, for example, to variations in user demand pattern. Motivated by the above, most of the existing works have focused on a specific time period within which a set of caches are pre-installed at certain nodes.

Broadly speaking, the schemes for caching content are classified into *pull-based* and *push-based* caching algorithms. Pull-based caching is a popular technique that is reactive and stores content in caches on-demand. Examples include the Least Frequently Used (LFU) and Least Recently Used (LRU) algorithms. On the other hand, push-based caching proactively estimates content requests and demand patterns, and preemptively store content to meet the user requests efficiently. Push-based caching algorithms have been proven to improve performance over pull-based caching techniques[27]. Therefore, in this thesis we focus on developing push-based caching algorithms.

1.3.1 Wired network caching

The caching problem has been well investigated in wired networks, with applications in content distribution networks (CDNs), peer-to-peer (P2P), information centric networks (ICNs) and internet protocol television (IPTV) networks. Unfortunately, this has been shown to be an NP-Hard problem in its general form, since it is closely related to the knapsack and facility location problems. Therefore, previous research efforts have focused on designing *approximation* algorithms that can provide solutions with performance guarantees or identifying special cases that are tractable. Baev et al. [28] investigated a cost minimization version of the caching problem, where transmitting content between caches incurs a cost. Under the assumption that costs form a metric, the authors presented a 10-approximation algorithm by rounding the optimal solution to a natural LP-relaxation of the caching problem. For the (equivalent) cost savings maximization version of the caching problem and without any restriction on costs, Borst et al. [18] presented a 2-approximation algorithm that iteratively swaps files in and out of the caches. For the special case of hierarchical networks and assuming that the costs between the caches form an ultra-metric, Korupolu et al. [29] presented a polynomial-time optimal caching algorithm by reduction to the minimum-cost flow problem. This generalizes the results presented in [30] where all costs are equal. Another optimal algorithm

was presented in [31] considering hierarchies of two levels with two caches installed at the bottom level. An analogy between the front-end request nodes and the back-end caches in a content distribution network with the input and output nodes of a switch was made in [34]. Here, queues of requests for different files build up at the request nodes, which route these requests to caches. A version of the well known max-weight scheduling algorithm was used for joint content placement and request routing, ensuring throughput optimality. Nevertheless, given the simple switch topology, routing is reduced to cache node selection (one-hop), and hence these techniques cannot be used in more general networks. When bandwidth limits are reached, e.g., in populated areas or during peak traffic hours, the caching policy needs to be jointly designed with the routing policy, that routes user requests to the caches [32], [17].

1.3.2 Wireless network caching

The caching problem obtains an interesting new twist in wireless networks. Namely, in the emerging hyper-dense heterogeneous cellular environments, the base stations may have overlapping coverage areas, which implies that the content can be delivered to users through multiple paths. Also, unlike the cache-nodes in wired networks, base station caches are typically not connected each other, and hence they can not exchange content. More importantly, novel challenges arise due to the broadcast nature of the wireless medium, that allows multiple users to receive a content file through a common *multicast* stream, and the *mobility* of the users, who may rapidly associate to multiple base stations as they move in space. Existing works have studied the caching problem assuming that all the requests are served via unicast (point to point) transmissions and that the users are static-they are in fixed locations. Based on these assumptions, various models have been proposed facing the caching problem from an optimization [33], [8], [35], an information theoretic [36] and a game theoretic point of view [37], [38]. The results span a wide range of techniques, such as discrete/convex optimization, content-centric algorithms, coalition formation and matching games. The caching problem was reconsidered in [39] to handle the case of mobile users requesting videos with different quality requirements (e.g., spatial resolution). In this case, each video is encoded into multiple segments (called versions and layers), and caching decisions are taken per segment, rather than per video. The impact of caching on the energy consumption and backhaul usage for renewable energy powered small cell networks with limited battery capacity and backhaul bandwidth was investigated in [40]. A stochastic geometry based wireless caching model along with an optimal probabilistic caching algorithm were presented in [41]. A different model was presented in [42] that considers base stations with conflicting objectives, i.e., each maximizing the quality of experience of its connected users. A

mechanism that allocates some of the available base station bandwidth to serve the local users and trades the rest with other caches in an auction-game fashion was proposed. Another auction-game based caching mechanism that takes into consideration both the utilities of the base station owners and the mobile users was proposed in [43].

However, all the above works neglect the broadcast nature of the wireless medium when deriving the caching policy. This is important for two reasons; first because of the *interference* caused by simultaneous wireless transmissions which can significantly reduce content delivery rate, and second because of the opportunity of serving multiple users with common interests via a dedicated *multicast* channel. For the first issue, the work in [44] investigated ways for mitigating the interference caused when multiple cache-endowed base stations deliver content to their associated users. The caching policy was derived with concerns on the cooperative MIMO (CoMP) technique that can be used to transform the cross-link interference into spatial multiplexing gain. This is possible by sharing both real-time channel state information (CSI) and payload data among the concerned base stations. For the second issue, the optimal multicast scheduling policy for a given cache placement at a base station has been explored in [45]. A joint caching and multicast scheduling policy in cellular networks was presented in [46]. Here, users are equipped with caches in order to store in advance multicasted content and retrieve later when they need it. More recently, Maddah-Ali et al. [47] developed a joint caching and multicast scheduling mechanism in tree networks aiming at reducing the peak traffic rate for serving a set of users, each one requesting a single file. However, the idea of caching content in base stations with concerns on the multicast schedule was firstly proposed in our recent work in [25].

Taking into consideration *mobility* profiles of the users is crucial to extract maximum benefit in network performance. This has been explored in wired networks where mobile users randomly connect to the leaf nodes in a cache hierarchy [48]. In wireless networks, the same problem has been studied by Guan et al. in [49]. The authors assume that the *exact trajectories* are known a priori for a set of moving users and optimize base station caching based on them. This seems to be an *over-optimistic* view of current mobility prediction mechanisms. In contrast, caching mechanisms that take as input the *probabilities* of user movements from one location to another appear to be a more realistic assumption. In our prior work [26], we have designed such mechanisms and demonstrated the performance benefits over conventional (mobility-agnostic) caching schemes.

Chapter 2

Hierarchical Caching

Contents

2.1 Introduction

Hierarchical topologies have been applied in many systems that provide massive content delivery services, such as IPTV and video on Demand (VoD), and gain increasing interest. In this chapter, we address the caching problem in multiple-level hierarchical systems, where caches may be installed at more than one levels. Requests for content files are generated at the bottom-level nodes of the hierarchy, which we refer to as *leaves*, and are routed upwards until they reach a cache-node that stores the requested file. If none of the accessed nodes has stored the respective file, a distant content server is triggered to serve the request. Since the latter option raises scalability concerns during peak usage hours and incurs high content delivery delay, our goal is to reduce the load of the server by serving as many requests as possible by the caches. Part of the results is also published in [19].

A key challenge in these systems is to devise the optimal caching policy: for a given anticipated content demand, determine which files should be placed in each cache, so as to maximize the number of requests served by the caches. Despite the plethora of work in this field, the optimal solution to the caching problem in multiple-level hierarchies remains unexplored. Existing optimal solutions are limited to hierarchies involving caches at the leaf nodes only [29], [30], being able though to serve one the requests of the others, associating a cost for each data transfer. However, installing caches at multiple levels differentiates the caching problem and calls for alternative solution techniques. The currently best performing methods achieve an approximation ratio close to 2, i.e., in the worst case half of the optimal number of requests are served by the caches [18], [33]. At present time, it is questionable *what is the computational complexity of the caching problem in multiple-level hierarchies and whether additional solutions with improved approximation guarantees are possible.*

Motivated by the above, we introduce a general optimization problem for devising the optimal caching policy for caches installed at multiple levels of hierarchies. This is an *integer optimization problem* and, thus, it is challenging to solve. We allow different leaf nodes to receive requests for different files with different intensity. The availability of the content at the caches determines whether the request will reach content servers or not. We show that the problem is NP-Hard in its general form by reduction from a variant of the set cover problem [50], and that it can be solved optimally in polynomial time when caches are installed only on a single hierarchy path. As the main contribution, we present an efficient algorithm with provably better approximation ratio than the best known counterparts for general cache hierarchies. Our methodology is based on expressing the caching problem as a maximization of a submodular function subject to uniform matroid constraints [51].

Our technical contributions can be summarized as follows:

- *Hierarchical Caching Problem (HCP).* We introduce the HCP problem that derives caching policies in multiple-level hierarchies. This is very important since hierarchical topologies have been applied in many systems that provide massive content delivery services, such as IPTV and video on Demand (VoD), and gain increasing interest.

- *Complexity of HCP problem.* We show that the HCP problem is NP-Hard in its general form by reduction from a variant of the set cover problem. This is a novel result considering that the NP-Hardness of the caching problem has been shown for arbitrarily-shaped networks [33], [28], not necessarily implying that the same statement holds when restricted to hierarchical topologies. For the special case that

caches are installed only on a single hierarchy path, we show that the constraint set of the HCP problem satisfies the total unimodularity property [52]. Therefore, the optimal solution can be obtained by solving the corresponding linear relaxed problem.

- *Approximation algorithms.* We express the HCP problem in its general form as a maximization of a submodular function subject to uniform matroid constraints [51]. This enables the derivation of a simple greedy algorithm with approximation guarantees that are provably better than those known.

- *Performance evaluation.* We evaluate the performance of the proposed algorithm for typical popularity distributions and demonstrate significant performance improvements (up to 56%) compared to conventional caching algorithms. The gains are higher when the popularity distribution is steep and the cache capacities at the upper hierarchy levels are large.

The rest of the chapter is organized as follows: Section 2.2 describes the system model and defines the HCP problem formally. Sections 2.3 and 2.4 present the complexity results and the approximation algorithms respectively. In Section 2.5 we present our numerical results.

2.2 System model and problem formulation

In this section we introduce the system model and formally define the hierarchical caching optimization problem.

System Model. We consider a general multiple-level hierarchical network like the one depicted in Figure 2.1. In an IPTV service system, access to the content servers is provided by the VHO (Video Head-end Office) for the users in a metropolitan area. Typically, content passes through a number of nodes such as intermediate offices (IO), central offices (CO), and digital subscriber line access multiplexer (DSLAM) before reaching a user. The operator may have installed caches at various nodes, possibly in more than one levels. We denote with $\mathcal{N}$ the set of nodes in the hierarchy, and with $C_n \geq 0$ (bytes) the size of the cache at node $n \in \mathcal{N}$.

We study the system for a certain time period (e.g., several days), during which the average demand for a set $\mathcal{F}$ of F popular files is assumed to be known in advance, as in [17], [18], [21], [33]. For example, the demand can be learned through analysis of previous time period statistics [53], [54], [55]. For notational convenience, we assume that all files have the same size, normalized to 1. This assumption can be easily removed as, in

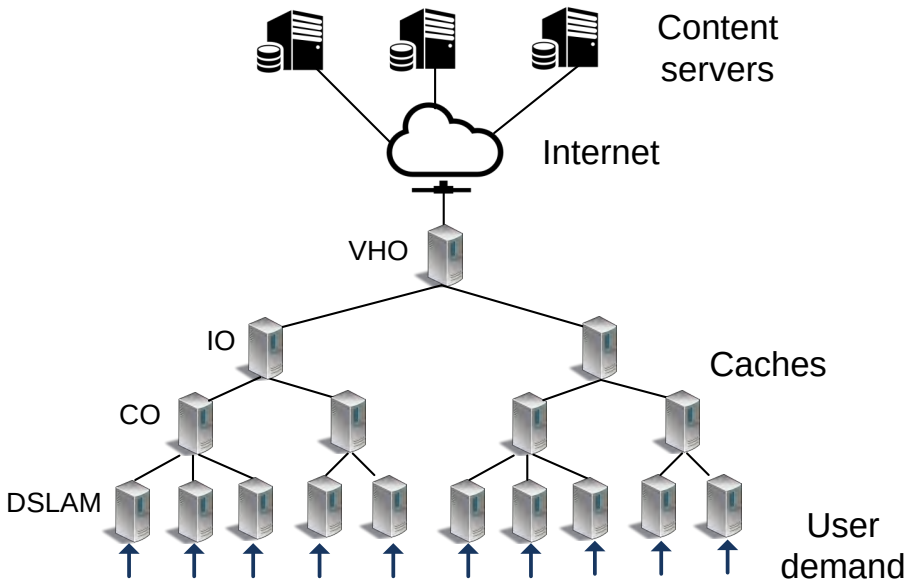


FIGURE 2.1: Graphical illustration of a hierarchical caching network.

real systems, files can be divided into blocks of the same length for convenience [17], [33]. User requests for content are generated at the bottom level nodes of the hierarchy, e.g., the DSLAMs, also called the *leaves*. We denote with $\mathcal{L} \subseteq \mathcal{N}$ the respective set of leaves, and with $\lambda_{nf} \geq 0$ the average user demand for file f generated at leaf n. The vector λ_n contains the demand values for a particular leaf n.

Let us also introduce the notation $\mathcal{P}_n$ to indicate the unique path uniting the leaf $n \in \mathcal{L}$ with the VHO (including the two endpoints). Each time a user request at leaf n is generated, it is served by the local leaf if it has the requested file cached. Otherwise, the request is routed upwards following the $\mathcal{P}_n$ path. The content servers are triggered to serve the request if none of the nodes on $\mathcal{P}_n$ path has the requested file cached. This latter option implies a significant server load and scalability concerns during peak usage hours. This exactly is the goal of this work: *"To carefully design the caching policy so as to effectively reduce the load of the content servers"*.

Problem Formulation. Let us introduce the integer decision variable $x_{nf} \in \{0, 1\}$ that indicates whether file $f \in \mathcal{F}$ is placed at cache-node $n \in \mathcal{N}$ ($x_{nf} = 1$) or not ($x_{nf} = 0$) [1]. We also define the respective caching policy matrix:

$$x = (x_{nf} \in \{0, 1\} : n \in \mathcal{N}, f \in \mathcal{F}) \tag{2.1}$$

[1]Note that we do not consider probabilistic content placement [41] or placement of encoded file portions [33] which are problems with continuous optimization variables. Instead, we tackle the (more challenging) binary-nature caching problem in this work.

The problem of determining the caching policy that minimizes the number of requests served by the content servers (server load) can be expressed as follows:

$$\min_{x} \quad \sum_{n\in\mathcal{L}}\sum_{f\in\mathcal{F}} \lambda_{nf} 1_{\left\{\sum_{n'\in\mathcal{P}_n}(x_{n'f})<1\right\}} \tag{2.2}$$

$$s.t. \quad \sum_{f\in\mathcal{F}}(x_{nf}) \leq C_n, \ \forall n \in \mathcal{N} \tag{2.3}$$

$$x_{nf} \in \{0,1\}, \ \forall n \in \mathcal{N}, \ f \in \mathcal{F} \tag{2.4}$$

where $1_{\{.\}}$ is the indicator function, i.e., $1_{\{c\}} = 1$ if condition c is true; otherwise $1_{\{c\}} = 0$. The expression in the objective function indicates that for each leaf node n that receives λ_{nf} requests for file f, the requests will reach the content servers if none of the caches on the path $\mathcal{P}_n$ has stored file f, i.e., when $\sum_{n'\in\mathcal{P}_n}(x_{n'f}) < 1$; otherwise the requests will be served by a cache on $\mathcal{P}_n$ that has stored this file. Inequalities in (2.3) denote the capacity constraints of the caches, whereas constraints in (2.4) indicate the discrete nature of the optimization variables.

One can note that it would be wasteful to place the same file more than once on the same path $\mathcal{P}_n$, for any $n \in \mathcal{L}$. Lemma 2.1 summarizes this point.

Lemma 2.1. *In the optimal caching policy, the files stored on $\mathcal{P}_n$ path are disjoint, i.e., no two nodes on $\mathcal{P}_n$ store the same file, $\forall n \in \mathcal{L}$.*

Lemma 2.1, the proof of which is deferred to Appendix A, is important since it limits the number of possible file placements in the caches and simplifies the problem. Based on it, we can formulate the (equivalent) problem of maximizing the number of requests served by the caches as follows:

$$\max_{x} \quad \sum_{n\in\mathcal{L}}\sum_{f\in\mathcal{F}} \lambda_{nf} \textstyle\sum_{n'\in\mathcal{P}_n} x_{nf} \tag{2.5}$$

$$s.t. \quad \text{constraints: (2.3), (2.4)}$$

$$\sum_{n'\in\mathcal{P}_n} x_{nf} \leq 1, \ \forall n \in \mathcal{L}, \ f \in \mathcal{F} \tag{2.6}$$

where constraint (2.6) is because of Lemma 2.1. Due to constraint (2.6) and the integrality of the optimization variables, the sum $\sum_{n'\in\mathcal{P}_n} x_{nf}$ can take either the value 1 or 0, $\forall n \in \mathcal{L}$, $f \in \mathcal{F}$. Hence, for each leaf n and file f, either all the λ_{nf} requests will be served by the caches ($\sum_{n'\in\mathcal{P}_n} x_{nf} = 1$) or none of them ($\sum_{n'\in\mathcal{P}_n} x_{nf} = 0$). We call the above the *Hierarchical Caching Problem* (HCP).

Although the objective function of the HCP problem is linear with respect to the optimization variables, the problem is non-trivial to solve due to its discrete nature. In the next two sections, we formally prove that HCP is NP-Hard in its general form, and

design optimal and approximation algorithms for the special case of caches installed on a single hierarchy path and the general case respectively.

2.3 Complexity of HCP problem

In this section we prove the high complexity of the HCP problem and identify a non-trivial special case where the problem can be optimally solved in polynomial-time.

2.3.1 Hardness of general case

Although the caching problem has been shown to be NP-Hard in general networks [28], [33], it remains questionable whether the same statement holds when restricted to hierarchical topologies. In this work, we answer this question in the affirmative by reduction from a variant of the set cover problem, which is NP-Hard [50]. In other words, we prove that the set cover variant is a special case of HCP, which implies that HCP is also NP-Hard. Particularly, the following theorem holds:

Theorem 2.2. *HCP is an NP-Hard problem.*

In order to prove Theorem 2.2, we will consider the corresponding (and equivalent) decision problem, called Hierarchical Caching Decision Problem (HCDP). Specifically:

HCDP: Given a multiple-level hierarchy with a set $\mathcal{N}$ of N nodes, a set $\mathcal{L} \subseteq \mathcal{N}$ of L leaves and their paths $\mathcal{P}_n$ to servers, a set $\mathcal{F}$ of F files, the cache sizes C_n $\forall n \in \mathcal{N}$, the user requests λ_{nf} $\forall n \in \mathcal{L}, f \in \mathcal{F}$, and a real number $Q \geq 0$, *we ask the following question*: does there exist a caching policy x, such that the value of the objective function in (2.5) is more or equal to Q and constraints (2.3),(2.4),(2.6) are satisfied?

We also consider the following variant of the set cover problem:

3-SCP [50]: Given a set of elements $\mathcal{U}$ (called the universe) and a family $\mathcal{S}$ of subsets of $\mathcal{U}$, a cover is a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of subsets whose union is $\mathcal{U}$. In the considered 3-SCP variant, the cardinality of each subset is at most three and the number of occurrences of each element in the subsets is exactly two. *We ask the following question*: given an integer k, does there exist a set cover of size k or less?

Lemma 2.3. *3-SCP problem is polynomial-time reducible to the HCDP.*

Proof. Given any instance of the 3-SCP problem, described by the $\mathcal{U}$, $\mathcal{S}$ and k values, we construct the equivalent instance of the HCDP problem as follows. There are caches

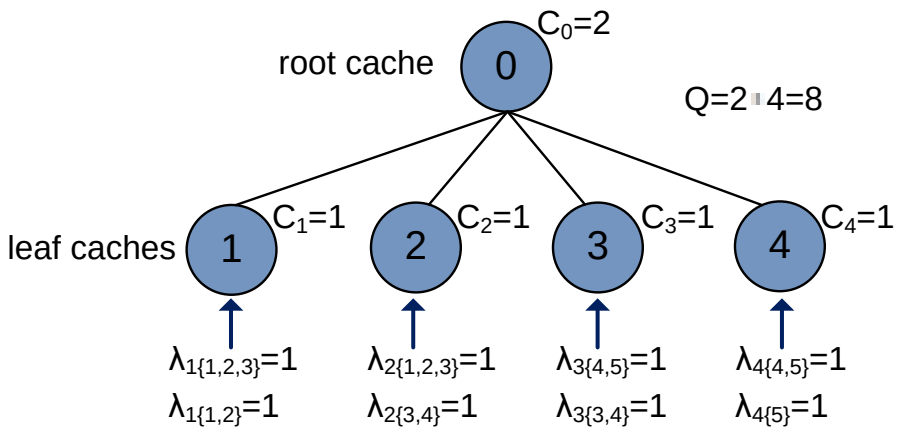


FIGURE 2.2: An example of the reduction from 3-SCP with $\mathcal{U} = \{1, 2, 3, 4, 5\}$, $\mathcal{S} = \{\{1, 2, 3\}, \{4, 5\}, \{1, 2\}, \{3, 4\}, \{5\}\}$ and $k = 2$.

at two levels of the hierarchy; a root cache and multiple leaf caches. For each subset $s \in \mathcal{S}$, we create a content file, which we refer to as f_s. Hence, there will be $F = |\mathcal{S}|$ files in total. For each pair of subsets s_1 and s_2 in $\mathcal{S}$ that overlap, i.e., they share at least one common element, we create a distinct leaf cache, denoted with $n_{s_1 s_2}$. Further, we set the demand at the leaf $n_{s_1 s_2}$ to be $\lambda_{n_{s_1 s_2} f_{s_1}} = \lambda_{n_{s_1 s_2} f_{s_2}} = 1$; zero for the rest files. Hence, the total number of file requests will be two times that of leaf caches, i.e., $\sum_{n \in \mathcal{L}} \sum_{f \in \mathcal{F}} \lambda_{nf} = 2L$. The size of each leaf cache is 1, whereas we set the root cache size to be equal to k. The question is whether there exists a caching policy that serves all the file requests by the caches, i.e., $Q = 2L$.

If the caches serve none of the requests, then the HCDP problem has a value of 0 (the worst case scenario). For each leaf node that the network manages to serve its requests completely through caching, the HCDP value increases by 2. This reduction is ensured only if the two files requested by that leaf are cached in the local leaf cache and/or the root cache. Notice though that each leaf cache can store up to one file, which implies that it can serve locally at most one of its two requests. Therefore, in order to achieve the desirable value $Q = 2L$, the root node needs to serve at least one request from each leaf node. That is, to store k files in the root such that each leaf node performs at least one request for these files. Since the two files requested by a leaf node correspond to two subsets that share common elements, and the number of occurrences of each element in the subsets is exactly two, finding k such files is equivalent of finding k subsets whose union is the universe of all elements. Hence, the HCDP problem is equivalent to the 3-SCP problem.

Figure 2.2 illustrates an example of the reduction from 3-SCP with $\mathcal{U} = \{1, 2, 3, 4, 5\}$, $\mathcal{S} =\{ \{1, 2, 3\}, \{4, 5\}, \{1, 2\}, \{3, 4\}, \{5\} \}$ and $k = 2$. In the HCDP instance there are $F = |\mathcal{S}| = 5$ files, a root cache of size $k = 2$ and $L = 4$ leaf caches, each of size 1. Each

leaf receives requests for two files. There is a solution to HCDP of value $Q = 2L = 8$ that places the files corresponding to subsets $\{1, 2, 3\}$ and $\{4, 5\}$ in the root cache, and the files corresponding to subsets $\{1, 2\}$, $\{3, 4\}$, $\{3, 4\}$ and $\{5\}$ in the leaf caches 1, 2, 3 and 4 respectively. Accordingly, the solution to 3-SCP picks the subsets $\{1, 2, 3\}$ and $\{4, 5\}$. □

The 3-SCP problem with the aforementioned restrictions on the cardinality of the subsets and the number of occurrences of the elements has been shown to be NP-Hard in [50], which completes the proof of Theorem 2.2.

2.3.2 Special case: caches installed on a single hierarchy path

Although the HCP problem is NP-Hard in its general form, in this subsection we show that it can be optimally solved in polynomial-time for a non-trivial special case. Specifically, we consider the network illustrated in Figure 2.1, but assume that there are caches installed only on a single hierarchy path, say $\mathcal{P}_{l^*}$, as in Figure 2.3(a). In this case, caching decisions are taken only for the nodes on this path:

$$x = (x_{nf} \in \{0, 1\} : n \in \mathcal{P}_{l^*}, f \in \mathcal{F}) \tag{2.7}$$

Hence, we can reformulate the HCP problem as follows:

$$\max_{x} \quad \sum_{n \in \mathcal{L}} \sum_{f \in \mathcal{F}} \lambda_{nf} \sum_{n' \in \mathcal{P}_n \cap \mathcal{P}_{l^*}} x_{nf} \tag{2.8}$$

$$s.t. \quad \sum_{f \in \mathcal{F}} (x_{nf}) \leq C_n, \ \forall n \in \mathcal{P}_{l^*} \tag{2.9}$$

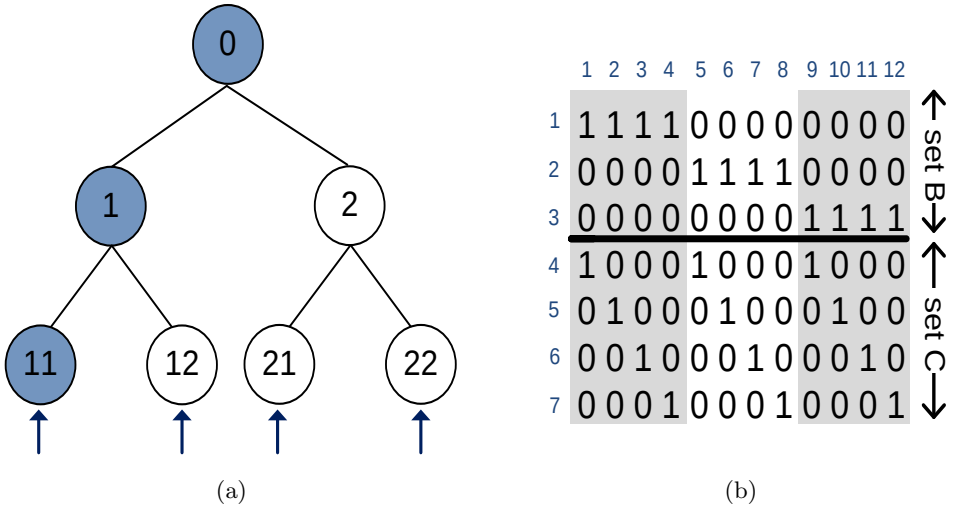
$$\sum_{n \in \mathcal{P}_{l^*}} (x_{nf}) \leq 1, \ \forall f \in \mathcal{F} \tag{2.10}$$

$$x_{nf} \in \{0, 1\}, \ \forall n \in \mathcal{P}_{l^*}, \ f \in \mathcal{F} \tag{2.11}$$

In order to show that the above problem is tractable, we will show that the integrality constraints in (2.11) are redundant; they can be replaced with the following constraints without changing the optimal solution:

$$x_{nf} \in [0, 1], \ \forall n \in \mathcal{P}_{l^*}, f \in \mathcal{F} \tag{2.12}$$

Hence, the optimal solution can be efficiently attained using standard linear optimization techniques [56] and software toolboxes like CPLEX and Mosek [57]. The proof is based on the *total unimodularity* property of the constraint matrix. Specifically, for a matrix A the following definitions and results hold [58]:



(a) (b)

FIGURE 2.3: (a) An example for a network with caches installed on a single path (solid circles), and (b) the respective constraint matrix described in (2.9),(2.10), for $F = 4$ files.

Definition 2.4. An integral matrix A is totally unimodular if the determinant of every square submatrix is 0, +1, or -1.

Proposition 2.5. *If for a linear program* $\{max\ c^T x\ :\ Ax \leq b\}$, *$A$ is totally unimodular and b is integral, then there is an optimal solution to the linear program that is integral.*

Note that the constraints in (2.9) and (2.10), namely $\sum_{f\in\mathcal{F}}(x_{nf}) \leq C_n,\ \forall n \in \mathcal{P}_{l^*}$ and $\sum_{n\in\mathcal{P}_{l^*}}(x_{nf}) \leq 1,\ \forall f \in \mathcal{F}$, can be written in the form $Ax \leq b$, where A and b are integrals. Hence, it suffices to show that A is totally unimodular in order for our problem to be solvable in polynomial time. To prove the total unimodularity property of matrix A, we use the following proposition [52]:

Proposition 2.6. *Let A be a matrix whose rows can be partitioned into two disjoint sets B and C. Then the following four conditions together are sufficient for A to be totally unimodular: (i) every column of A contains at most two non-zero entries, (ii) every entry in A is 0, +1, or -1, (iii) if two non-zero entries in a column of A have the same sign, then the row of one is in B, and the other in C, and (iv) if two non-zero entries in a column of A have opposite signs, then the rows of both are in B, or both in C.*

In our case, every column contains exactly two non-zero elements, each with value +1. Hence, conditions (i), (ii) and (iv) are satisfied. To show that condition (iii) is satisfied, we note that each column includes a non-zero element in a row corresponding to constraint (2.9) and another in a row corresponding to constraint (2.10). Hence, we can

partition the rows of matrix A into a set B containing the rows in constraint (2.9) and a set C containing the rows in constraint (2.10), satisfying condition (iii). Figure 2.3(b) depicts an example of the constraint matrix. Since proposition 2.6 is satisfied, we obtain the following lemma:

Lemma 2.7. *The constraint matrix described in inequalities (2.9),(2.10) is totally unimodular.*

Hence, we obtain the following theorem:

Theorem 2.8. *When caches are installed on a single hierarchy path, the optimal caching policy can be found in polynomial time.*

2.4 Approximation algorithms

In this section, we investigate the hierarchical caching problem in its general form and derive algorithms with improved approximation guarantees compared to state-of-the-art methods [18],[33]. We start by presenting a simple greedy algorithm achieving 1.582-approximation in two-level hierarchies. Then, we show how it can be extended to handle the general case of any number of levels.

2.4.1 An 1.582-approximation algorithm for two-level hierarchies

We consider a two-level hierarchy, as in Figure 2.2, with a root cache, indexed by 0, and L leaf caches, indexed by 1, 2, ..., L. As we showed in Theorem 2.2, the respective caching problem is NP-Hard. Hence, exact solution approaches are not practical and the use of approximation algorithms is justified. Subsequently, we derive such an approximation by expressing the caching problem as a maximization of a submodular function subject to uniform matroid constraints [51]. This is a novel result that is specific to the hierarchical structure of the underlying topology, and does not hold in other types of networks[2]. We begin by introducing the definition of submodular functions.

Definition 2.9. Given a finite set of elements G, a function $h : 2^{\mathcal{G}} \rightarrow \mathbb{R}$ is submodular if for every sets $X \subseteq Y \subseteq G$ and every element $g \in G \setminus Y$, it holds that:

$$h(X \cup \{g\}) - h(X) \geq h(Y \cup \{g\}) - h(Y) \tag{2.13}$$

[2]For example, in [33] and [35], the constraints of the caching problems were expressed as *partition* rather than *uniform* matroids.

The set G is often referred to as *ground set*. The submodularity property specifies that the marginal value of the function when adding a new element in a set decreases as this set becomes larger.

For a given file placement in the root cache, the optimal file placement in the leaf caches can be efficiently computed; every leaf n stores the C_n most popular files with respect to λ_n, but the files in the root (cf. Lemma 2.1). Let us denote the placement of file f in the root cache by an element e_f and define the ground set G consisting of all elements as:

$$G = (e_f : f \in \mathcal{F}) \tag{2.14}$$

Then, every possible caching solution can be expressed by a subset $X \subseteq G$, where the elements included in X correspond to the files placed in the root cache. Due to the cache capacity limitation of the root, it should be $X \subseteq I$ where:

$$I = \{X \subseteq G : |X| \leq C_0\} \tag{2.15}$$

The pair (G, I) defines a *uniform matroid* [51].

Based on the above, we can write the objective function in (2.5) as a function of the set X:

$$h(X) = \sum_{n \in \mathcal{L}} \left(\sum_{f \in \mathcal{F}: e_f \in X} (\lambda_{nf}) + \sum_{f \in M_n(X)} (\lambda_{nf}) \right) \tag{2.16}$$

where $M_n(X)$ denotes the files placed in the leaf cache n given the file placement in the root X. The first term in the sum corresponds to the requests served by the root, whereas the second to the requests served by the leaf caches.

Then, we have the following lemma:

Lemma 2.10. *The function $h(X)$ is monotone, increasing and submodular.*

Proof. Since the sum of submodular functions is also submodular, it suffices to show that every term of the external sum in (2.16) is a submodular function. Let us focus on a single leaf cache $n \in \mathcal{L}$, and consider adding element e_f in a set X. We distinguish the following two cases: **(i)** If file f is not included in $M_n(X)$, then storing file f in the root cache enables the requests generated at leaf n to be served by the root cache instead of the content servers, resulting a marginal value of $h(X \cup \{e_f\}) - h(X) = \lambda_{nf}$. **(ii)** Otherwise, storing file f in the root cache forces leaf cache n to swap file f with the most popular file with respect to λ_n but those files already cached in the root and the local leaf cache. Let us denote with f' that file. Then, the marginal value will be $h(X \cup \{e_f\}) - h(X) = \lambda_{nf'}$.

Algorithm 2.1: Greedy algorithm for two-level hierarchies

$X \leftarrow \emptyset$
for *iteration* $= 1, 2, ..., C_0$ **do**
  $e_f^* \leftarrow \text{argmax}_{e_f \in G \setminus X}\{h(X \cup \{e_f\}) - h(X)\}$
  $X \leftarrow X \cup \{e_f^*\}$
Caching is done according to X for the root and $M_n(X)$ for every leaf $n \in \mathcal{L}$.

We now consider adding the same element e_f in a set $Y \supseteq X$. We distinguish the following two cases: **(i)** If file f is not included in $M_n(Y)$, then, by definition of the $M_n(.)$ set, file f is not included in $M_n(X)$ neither. Hence, the marginal value will be $h(Y \cup \{e_f\}) - h(Y) = h(X \cup \{e_f\}) - h(X) = \lambda_{nf}$. **(ii)** If file f is included in $M_n(Y)$, then, the marginal value is $h(Y \cup \{e_f\}) - h(Y) = \lambda_{nf''}$, where f'' is the the most popular file with respect to λ_n but those files already cached in the root and the local leaf cache. We distinguish the following two subcases: **(ii.a)** File f is not included in $M_n(X)$. Then, $h(X \cup \{e_f\}) - h(X) = \lambda_{nf} \geq \lambda_{nf''}$. **(ii.b)** File f is included in $M_n(X)$. Then, $h(X \cup e_f) - h(X) = \lambda_{nf'} \geq \lambda_{nf''}$, where the last inequality is because file f'' is picked among a subset of the files used for picking f'.

Hence, the marginal value for adding an element in Y is always lower or equal to the one in X, which implies that $h(.)$ is submodular. Finally, it is not hard to show that as more files are stored in the root cache, more requests are served by the caches. Hence, $h(Y) \geq h(X)$, which implies that $h(.)$ is monotone and increasing. □

A greedy algorithm obtains an approximate solution for the problem of maximizing a submodular function subject to uniform matroid constraints, with a performance that is provably at most $e/(e-1) = 1.582$ times worse than optimal [51]. The algorithm starts with an empty set, and at each iteration it adds the element with the highest marginal value to the set, while satisfying inequality (2.15). The procedure is summarized in Algorithm 2.1.

Algorithm 2.1 runs in C_0 iterations. At each iteration, it computes the value $h(X \cup \{e_f\})$ for each one of at most F elements in order to determine e_f^*. Each one of these computations requires finding the $M_n(X \cup \{e_f\})$ set for every leaf $n \in \mathcal{L}$, i.e, the C_n most popular files with respect to λ_n, but those corresponding to $X \cup \{e_f\}$. Assuming that the λ_n vector is initially sorted, these files can be found by traversing λ_n until C_n such values are obtained. In the worst case, the $C_n + C_0$ first elements will be traversed. Hence, the overall complexity for all iterations is upper bounded by: $C_0 F \sum_{n \in \mathcal{L}}(C_n + C_0)$. Since sorting a vector of size F requires $F \log F$ time [59], we obtain the following theorem:

Theorem 2.11. *In a two-level hierarchy, Algorithm 2.1 finds a 1.582-approximate solution to the caching problem in* $LF \log F + C_0 F \sum_{n \in \mathcal{L}}(C_n + C_0)$ *time.*

This improves the $(2L-1)/L$-approximation ratio for two-level hierarchies given in [18], as well as the 2-approximation ratio given in [33]. Also, for the special case of $L = 2$ leaves, we can show that Algorithm 2.1 finds the optimal solution.

Theorem 2.12. *In a two-level hierarchy with two leaves, Algorithm 2.1 finds the optimal solution to the caching problem in* $2F \log F + C_0F(C_1 + C_2 + 2C_0)$ *time.*

The proof of Theorem 2.12 is deferred to Appendix B.

2.4.2 Extension to multiple-level hierarchies

In this subsection, we show how to extend Algorithm 2.1 for general (not necessarily two-level) hierarchies. We start with a three-level hierarchy, as the one depicted in Figure 2.4. Using the same arguments as in the previous subsection, we can express the objective function in (2.5) as a submodular function of the files placed in the root cache. However, in contrast to the two-level case, here, for a given file placement in the root cache X, we can not efficiently compute the optimal file placement in the rest caches. Namely, for each direct descendant of the root cache, e.g., caches 1, 2 and 3 in Figure 2.4, there is a two-level hierarchy subproblem involving this cache and its descendants. According to Lemma 2.3, each one of these subproblems is NP-Hard by itself. Hence, the best we can get is an α-*approximate* solution to each subproblem by applying Algorithm 2.1. We note, however, that when solving these subproblems, we need to ensure that the files in X will not be placed in any other cache (cf. Lemma 1).

With that in mind, we can extend Algorithm 2.1 to determine the file placement in the root cache of a three-level hierarchy. As in the two-level case, we iteratively choose the file to be placed in the root cache based on its marginal value. However, since we can not efficiently find the choice with the highest marginal value (since we can not solve optimally the two-level subproblems), at each one of the C_0 iterations we pick an element with a marginal value that is at most α times worse than that of the optimal choice. This is possible by applying Algorithm 2.1 to approximately solve each one of the two-level subproblems formed, one time for each possible file choice in the root, and picking the file that results the highest marginal value. Once the file placement in the root cache is found, we decide the file placement in the rest caches by applying Algorithm 2.1, one time for each subproblem, ensuring that the files placed in the root cache will not be be placed in any other cache.

When an α-approximation algorithm is used to find the element with the best marginal value to add in the greedy solution, the greedy algorithm outputs a $(e^{1/a}/(e^{1/a}-1))$-approximate solution [60]. This generalizes the results presented for two-level hierarchies

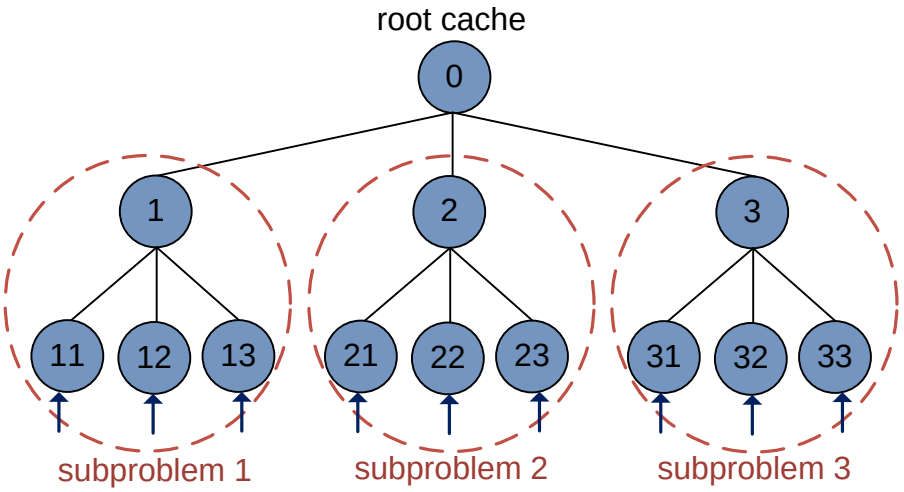


FIGURE 2.4: Graphical illustration of the application of Algorithm 2.1 to a three-level cache hierarchy.

(where $\alpha = 1$), to multiple-level hierarchies. Specifically, for three-level hierarchies, we have $\alpha = 1.582$, which yields an $e^{1/1.582}/(e^{1/1.582} - 1) = 2.1343$ approximation ratio for the greedy algorithm. Similarly, we can use the 2.1343-approximation algorithm for three-level hierarchies to obtain an approximation ratio of $e^{1/2.1343}/(e^{1/2.1343} - 1) = 2.6732$ for four-level hierarchies and so on.

In the next section, we numerically show the performance benefits of Algorithm 2.1 over the state-of-the-art methods in [18],[33].

2.5 Performance evaluation

In this section, we present the numerical results of the experiments that we have conducted to show the superiority of the proposed caching scheme over certain commonly used schemes. Specifically, we compare the performance of the following four schemes:

1. *Greedy* [18]: Each leaf cache stores the most popular files with respect to its local demand. Then, moving from the bottom to the top-level, each cache stores the most popular files with respect to the demand that was left unserved by its descendants.

2. *Swapping* [18]: It starts with a random cache placement. Iteratively, it swaps a file that is currently included in a cache to one not included in it if this increases the requests served by the caches, i.e., $\sum_{n\in\mathcal{L}}\sum_{f\in\mathcal{F}}\lambda_{nf}\sum_{n'\in\mathcal{P}_n}x_{nf}$. The process is iterated until no possible swap can improve performance.

3. *Femtocaching* [33]: It starts with all the caches being empty. Iteratively, it performs the placement of a file to a cache that maximizes the requests served by the caches, i.e., $\sum_{n\in\mathcal{L}}\sum_{f\in\mathcal{F}}\lambda_{nf}\sum_{n'\in\mathcal{P}_n}x_{nf}$. The procedure terminates when all the caches become full.

4. *Algorithm 2.1*: The proposed scheme in Algorithm 2.1 extended to multiple-level hierarchies.

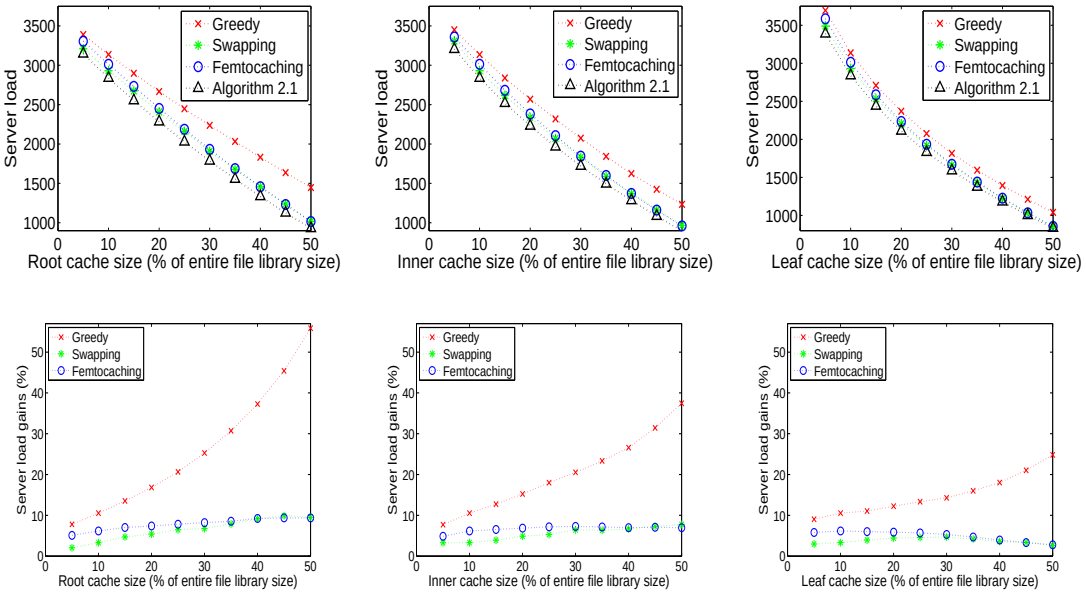
The performance criterion that we consider is the total number of requests that reach the serves (*server load*), which is defined by expression (2.2). To describe in detail the benefits of the proposed scheme, we also depict the normalized difference between the server load achieved by any of the first three schemes and the proposed one (*server load gains*). Formally, the server load gains for the Greedy algorithm are defined as:

$$\frac{server\_load_{Greedy} - server\_load_{Algorithm2.1}}{server\_load_{Algorithm2.1}} \tag{2.17}$$

where $server\_load_{scheme}$ denotes the server load achieved by the associated scheme. A similar definition holds for the Swapping and Femtocaching schemes.

Simulation Setup. We consider the three-level cache hierarchy depicted in Figure 2.4, consisting of a single root cache (indexed by 0), three inner caches (indexed by 1, 2, 3) and nine leaf caches (indexed by 11, 12, 13, 21, 22, 23, 31, 32, 33). We simulate the delivery of a library of $F = 500$ popular files, for which recurring requests are expected. Specifically, within the evaluation period each leaf node receives $1,000$ requests for these files, resulting to $9,000$ requests in total. Following empirical studies in VoD systems, we model content popularity using a Zipf distribution, i.e., the request rate for the i^{th} most popular file is proportional to i^{-z}, for some shape parameter $z > 0$ [61], [62]. In order to simulate diverse popularity distributions, the ranks of the files are randomly permuted in every leaf node. Unless otherwise specified, each cache is capable of storing 10% of the entire file library size, whereas we set $z = 0.8$ [62]. Throughout, we evaluate the performance of the four schemes for different values of the cache sizes per hierarchy level and the zipf shape parameter z. For the algorithms' implementation we used the C++ language in the Visual Studio environment.

Impact of cache sizes. We first compare the performance of the four schemes for different sizes of the caches. In the experiment in Figure 2.5(a), the size of the root cache spans a wide range of values, starting from 5% to 50% of the entire file library size, reflecting different operator conditions. As expected, increasing the root cache size reduces server load for all schemes, since more files become available for download within the paths to servers. The proposed scheme (Algorithm 2.1) performs markedly better than the other schemes, especially for large values of the root cache size. The



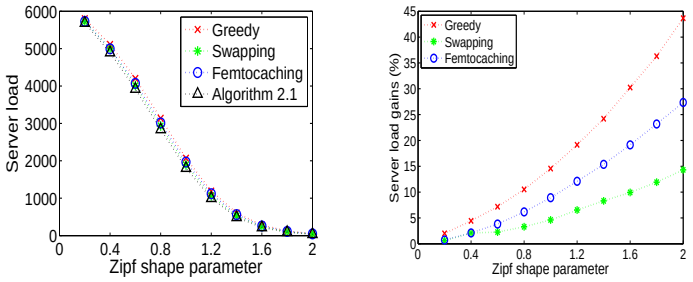
(a) Impact of root cache size (b) Impact of inner cache size (c) Impact of leaf cache size

FIGURE 2.5: Performance comparison between Greedy, Swapping, Femtocaching and Algorithm 2.1 for various values of (a) the root cache size, (b) the inner cache size and (c) the leaf cache size.

server load gains are up to 56%, 10% and 9.5% when compared to Greedy, Swapping and Femtocaching scheme respectively.

We repeat the above experiment, but vary the size of each one of the three inner caches rather than the root. The results are depicted in Figure 2.5(b). Although the shapes of the curves are similar to that in Figure 2.5(a), the server load gains are now lower, up to 37%, 8% and 7.5% when compared to Greedy, Swapping and Femtocaching scheme respectively. Finally, Figure 2.5(c) depicts the results when we vary the size of the leaf caches. In this case, the server load gains are limited to 25%, 5% and 6% over Greedy, Swapping and Femtocaching scheme respectively. Thus, we can infer that *the superiority of Algorithm 2.1 over the existing schemes is more pronounced for large sizes of the caches installed at the upper hierarchy levels.*

Impact of popularity distribution. Figures 2.6(a)-2.6(b) show the impact of the zipf shape parameter z on the performance of the four schemes. We observe that as the z value increases, the server load decreases for all the schemes, reflecting the well known fact that caching effectiveness improves as the popularity distribution gets steeper [18]. Interestingly, *Algorithm 2.1 consistently outperforms the other schemes, with the gains increasing as z increases.* Particularly, while the server load gains over Greedy, Swapping and Femtocaching scheme are 2%, 0.6% and 0.7% respectively when $z = 0.2$, they increase to 44%, 14.5% and 27.5% when $z = 2$.



(a) (b)

FIGURE 2.6: Performance comparison between Greedy, Swapping, Femtocaching and Algorithm 2.1 for various values of the zipf shape parameter.

Main takeaways: The hierarchical caching problem is NP-Hard even in two-level networks. Nevertheless, the problem can be optimally solved in polynomial-time when all then caches are installed on a single hierarchy path. The simple iterative algorithm that we propose (Algorithm 2.1) achieves a provably better approximation ratio than the best-known counterparts. In our simulation the performance gains are up to 56%, being more pronounced when the cache capacities at the upper hierarchy levels are large and the content popularity distribution is steep.

Chapter 3

Joint Caching and Routing in Wireless Networks

Contents

3.1 Introduction

In order to cope with the mobile data traffic explosion [1] mobile network operators (MNOs) deploy small cell base stations (SBSs) which operate in conjunction with the macrocell base stations (MBSs) [20]. This architecture benefits both the MNO by replacing the long-range costly transmissions of the MBSs, and the users by offering them high-capacity, energy-prudent communication links. However, the operation of these small cells presumes the existence of high-speed backhaul links connecting the deployed base stations with the core network. Decentralized caching architectures have been recently proposed [33], [8] with the goal to minimize peak traffic - and subsequently the cost - of these backhaul links. The main idea is to cache in advance (during off-peak demand) popular content items at the SBSs so as to reduce, especially during peak traffic hours, the requests that are routed over the backhaul links to the core network.

Given the vast set of the content items, the challenge in this context is to find the optimal caching policy. That is, decide which items should be cached at each base station, so as to maximize the portion of user requests that are satisfied locally by the SBSs, without using backhaul links or employing the MBSs. Unfortunately though, this has been proved to be in general an NP-hard problem [33]. The problem becomes even more challenging if one considers *massive* content delivery scenarios, e.g., in populated areas or during peak traffic hours. In these cases, mobile data delivery will be constrained by the transmission capacity of the SBSs. Obviously, in order to deliver a content item to a user, it does not suffice to have it cached at a base station within the user's transmission range, but additionally the base station should have enough capacity to deliver it. Prior works assume that the transmission capacity is rarely the bottleneck for the caching base stations. Clearly, this is not a realistic assumption for dense urban areas where user content demand is often massive.

In this chapter, we consider the scenario of massive content delivery through cache-endowed SBSs with *hard bandwidth constraints* that bottleneck the data transmission to mobile users. We consider the case that user *requests are unsplittable*, which implies that each one of the requests is entirely satisfied by one base station, i.e., a user that requests a file will not receive different *parts* of it from different base stations, but a complete replica from a single base station. This is of major importance, since associating a user request with multiple base stations incurs the extra effort to synchronize the communication. Besides, user association cannot change in a very small time scale as base station reselection requires a time interval of several seconds [63].

We introduce the *joint routing and caching for un-splittable requests* (JRC-UR) problem, the solution of which maximizes the content requests that are satisfied by the SBSs. We

propose a novel mapping of the JRC-UR problem to a variant of the facility location problem known as the *Unsplittable Hard-Capacitated Metric Facility Location Problem* (UHCMFL). The UHCMFL problem has been studied extensively, and there exist a variety of bi-criteria approximation algorithms for its solution [64]-[69]. A bi-criteria (α, β)-approximation algorithm ensures an α-approximation solution under the assumption that the facility capacities can be violated up to β times. We prove that the JRC-UR can be reduced in polynomial time to UHCMFL. Moreover, we provide a methodology for transforming the above bi-criteria facility location algorithms to approximation caching algorithms. We evaluate numerically one of the derived approximation algorithms in representative scenarios, and find that its performance (in terms of requests routed to MBS) is up to 38% better than conventional caching schemes.

Going one step further, we design joint caching and routing policies for caches owned by residential WiFi users. Since the residential users are self-interested, the MNO must offer them proper (monetary) compensation in order to agree to cache and deliver the requested content to mobile users. The offered reimbursements directly determine the available amounts of bandwidth and cache space in every AP, which in turn affect the caching policy and the routing policy. We show that this is a NP-Hard problem in its general form, and we derive a policy that minimizes the cost of the MNO using the Primal-dual method. Using a novel WiFi usage dataset collected from 167 residential users, we show that in densely-populated areas with costly network capacity upgrades, our proposal reduces operator's total cost by a factor of 2, while reimbursing up to 9 euros per month each residential user.

The rest of the chapter is organized as follows. Section 3.2 describes the system model and the assumptions, and introduces formally the problem. In Section 3.3 we reduce the problem to the UHCMFL problem. Section 3.4 presents an approximation framework based on the above reduction, whereas Section 3.5 provides the numerical results. The extension to the case of user-owned caches is described in 3.6.

3.2 System Model and Problem Formulation

In this section, we first introduce the system model and explain the considered network architecture. In the sequel, we provide a simple, yet representative, example that motivates the joint design of routing and caching policies, and we formally introduce the respective optimization problem.

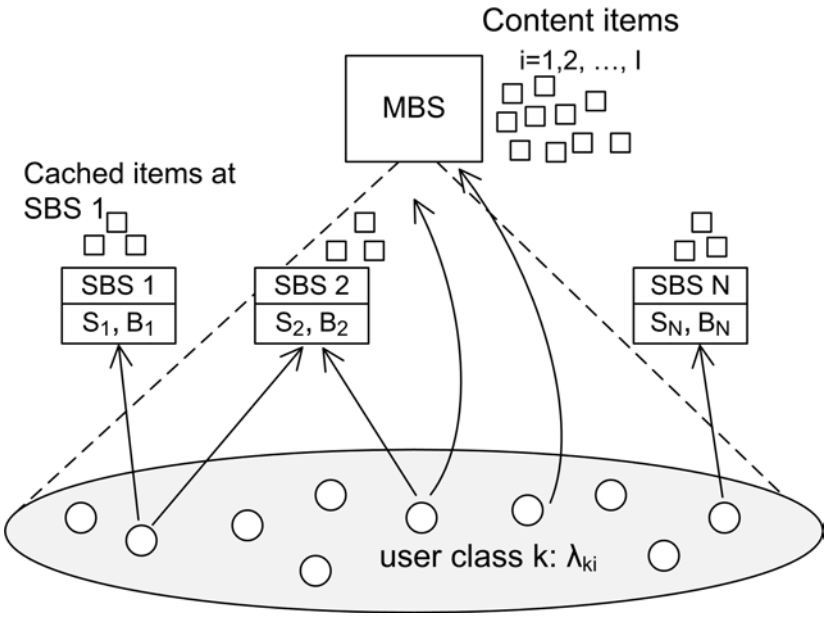


FIGURE 3.1: Mobile users are randomly distributed in the coverage regions of the SBSs. Each SBS n has certain storage and bandwidth capacity of S_n and B_n units respectively.

3.2.1 System Model

We consider a single macrocell[1] in which the mobile network operator (MNO) serves the content requests submitted by a set $\mathcal{K} = \{1, 2, \ldots, K\}$ of $K = |\mathcal{K}|$ classes of mobile users (MUs). Each class represents the users lying in a certain geographic area[2]. Hence, it is possible to have more than one requests for one file originating from the same point. Also, there exists a set $\mathcal{N} = \{1, 2, \ldots, N\}$ of $N = |\mathcal{N}|$ small cell base stations (SBSs) which operate in conjunction with the macrocell base station (MBS), yielding a heterogeneous cellular network. This two-layer architecture is depicted in Figure 3.1.

We consider the case that the SBSs operate in disjoint subchannels than the MBS. Also, we assume that neighboring SBSs are assigned orthogonal frequency bands and/or employ enhanced inter-cell interference coordination techniques (eICIC) proposed in LTE Rel. 10 [70]. We assume that time is slotted and we study the system for one time period T. Each SBS $n \in \mathcal{N}$ has a certain transmission capacity, i.e., it can deliver $B_n \geq 0$ data bytes within period T. Additionally, each SBS $n \in \mathcal{N}$ is endowed with a storage capacity of $S_n \geq 0$ bytes.

Let the set $\mathcal{I}$ indicate a static collection of $I = |\mathcal{I}|$ content items (or, files). For notational convenience, we assume that all files have the same size s. This assumption can be easily removed as, in real systems, files can be divided into blocks of the same length or by

[1] The study can be directly extended for more macrocells.
[2] Hereafter, we may use the term *user* $k \in \mathcal{K}$ to refer to *user class* $k \in \mathcal{K}$.

leveraging advanced coding techniques [33]. We denote with $\lambda_{ki} \in \mathcal{Z}^+$ the expected number of requests for file $i \in \mathcal{I}$ generated by user class $k \in \mathcal{K}$ within T. Observe that a user class consists of many users and thus can generate more than one requests for the same file. User requests may change over consecutive time periods but are considered fixed (and known) within each period. This is a realistic assumption for proactive caching as the popularity distribution of the files changes slowly [33]. The coverage areas of the SBSs are *overlapping*. Let $\mathcal{N}_k \subseteq \mathcal{N}$ denote the set of SBSs that are in communication range with user class k. Then, a request generated by k can be satisfied by any of the SBSs in $\mathcal{N}_k$ that owns a copy of the requested item. The requests that cannot be satisfied locally, i.e., by any SBS, are routed to the MBS. Our goal is to minimize this latter quantity which depends both on the caching and the routing policy of the operator.

3.2.2 Motivating Example

Consider the system depicted in Figure 3.2 with two SBSs (n_1 and n_2) and three users (k_1, k_2 and k_3). The circles represent the coverage areas of the SBSs while all the users are also covered by the MBS (not shown). There are also two equal-sized files denoted i_1 and i_2. Each SBS can cache at most one file due to its limited storage capacity. Also, because of the bandwidth limitations, n_1 can serve at most 5 requests and n_2 can serve at most 10 requests. User class k_1 requests file i_1 1 time, k_2 requests i_1 2 times and k_3 requests file i_2 10 times. The optimal routing and caching strategy is the one that maximizes the requests satisfied by n_1 and n_2. In this example, this policy dictates to cache i_1 to n_1, and i_2 to n_2. Then, n_1 serves the request for i_1 generated by k_1, and n_2 serves all the requests for i_2 generated by k_3. Hence, only 2 requests need to be served by the MBS.

However, if we omit the transmission capacity constraints, then the *optimal caching policy changes*: it places i_2 to n_1, and i_1 to n_2. Then, n_1 handles all the requests of k_3, and n_2 handles all the requests of k_2, letting only the one request generated by k_1 to be routed to the MBS. Nevertheless, in practice, n_1 will serve only half of incoming requests (due to limited capacity) and redirect the rest to the MBS. Hence, the MBS will have to serve $6 > 2$ requests.

This example demonstrated that *ignoring the SBSs' bandwidth capacities when designing the caching policy leads the system to inefficient operating points*, for the case of massive content requests where the capacity limits of the SBSs are reached. In the sequel, we formalize the respective problem for the general case.

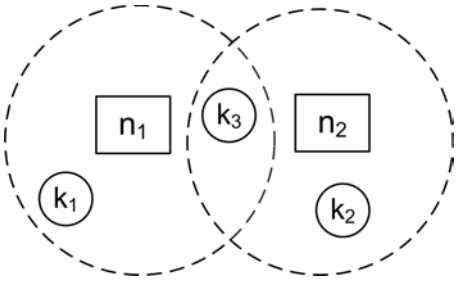


FIGURE 3.2: An example with 2 SBSs (n_1 and n_2) and 3 users (k_1, k_2, k_3). The circles denote the transmission range of each SBS.

3.2.3 Problem Formulation

Let us introduce the binary decision variable $x_{ni} \in \{0, 1\}$ which indicates whether file $i \in \mathcal{I}$ is placed at the cache of SBS $n \in \mathcal{N}$ or not. We also define the respective *caching policy* matrix:

$$x = (x_{ni} : n \in \mathcal{N},\, i \in \mathcal{I}). \tag{3.1}$$

Additionally, let the integer decision variable $y_{ni}^{k} \in \mathcal{Z}^{+}$ indicate the number of requests for file i generated by user class k that are routed to SBS n. Also, $y_{Mi}^{k} \in \mathcal{Z}^{+}$ denotes the number of requests for file i generated by user $k \in \mathcal{K}$ that are routed to the MBS which is denoted with M. The *routing policy* of the operator is described by the following matrix:

$$y = \left(y_{ni}^{k} : n \in \mathcal{N} \cup \{M\}, i \in \mathcal{I}, k \in \mathcal{K}\right). \tag{3.2}$$

Observe that each one of the λ_{ki} requests for file i generated by user class k must be satisfied by exactly one SBS, $\forall i \in \mathcal{I}, k \in \mathcal{K}$. This means that each variable y_{ni}^{k} is allowed to take value in the *integer set* $\{0, 1, ..., \lambda_{ki}\}$ ("un-splittable requests"). The above integrality constraint makes the problem even harder compared to the simplified case that any fraction of the total user demand for a file is allowed to be routed to multiple SBSs, i.e., the variable y_{ni}^{k} take values in the *real set* $[0, \lambda_{ki}]$ ("splittable requests") [17].

As we explained in the previous example, the routing policy should take into account the bandwidth capacity constraint B_n of each SBS $n \in \mathcal{N}$. Clearly, if an SBS is already congested, it cannot serve additional user requests. Moreover, routing decisions are coupled with the respective caching decisions: a request is routed to an SBS only if the latter has the requested content item cached. At the same time, the caching policy must respect the storage capacity S_n of each base station $n \in \mathcal{N}$.

Summarizing, the problem of devising the joint routing and caching policy for un-splittable requests (JRC-UR problem) which minimizes the requests routed to the MBS,

can be formulated as follows:

$$\min_{x,y} \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}} y_{Mi}^k \tag{3.3}$$

s.t.

$$\sum_{i\in\mathcal{I}} x_{ni} s \leq S_n,\ \forall n \in \mathcal{N}, \tag{3.4}$$

$$\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}} y_{ni}^k s \leq B_n,\ \forall n \in \mathcal{N}, \tag{3.5}$$

$$y_{ni}^k \leq x_{ni}\lambda_{ki},\ \forall i \in \mathcal{I}, k \in \mathcal{K}, n \in \mathcal{N}, \tag{3.6}$$

$$y_{ni}^k = 0,\ \forall i \in \mathcal{I},\ k \in \mathcal{K}, n \in \mathcal{N} \setminus \mathcal{N}_k, \tag{3.7}$$

$$\sum_{n\in\mathcal{N}\cup\{M\}} y_{ni}^k = \lambda_{ki},\ \forall i \in \mathcal{I}, k \in \mathcal{K}, \tag{3.8}$$

$$x_{ni} \in \{0,1\},\ \forall n \in \mathcal{N},\ i \in \mathcal{I}, \tag{3.9}$$

$$y_{ni}^k \in \mathcal{Z}^+,\ \forall n \in \mathcal{N}\cup\{M\}, i \in \mathcal{I},\ k \in \mathcal{K}, \tag{3.10}$$

where inequalities (3.6) indicate that SBSs can not serve requests for files that are not in their caches. Constraints (3.7) denote that SBSs can not serve requests generated by users located out of their coverage areas, and (3.8) dictate that the system must serve all the requests (inelastic demand[3]). Finally, (3.9)-(3.10) reveal the discrete nature of the optimization variables.

Clearly, the above problem is very hard to solve optimally. Namely, the following lemma holds.

Lemma 3.1. *The JRC-UR problem is NP-hard.*

Proof. The JRC-UR problem is a generalization of the *Helper Decision Problem* (HDP), described in [33], by incorporating the hard bandwidth constraints of the SBSs. Hence, problem HDP, which is NP-hard, can be directly reduced in polynomial time to our problem. Consequently, JRC-UR is also NP-hard. □

3.3 Reduction to Facility Location Problem

In this section our goal is to devise a polynomial time reduction of the JRC-UR problem to a well known facility location problem. This will help us in the sequel to derive

[3]Notice that we have not included a capacity constraint for the MBS, assuming that it can accommodate all the unsatisfied requests even if this entails a very high OpEx cost.

approximation algorithms for our problem, by using the ones that have been designed for the facility location problem. Note that although, in general, reduction preserves only optimality (and not approximation bounds), for our case it also holds that we can compute how much in the worst case the approximation ratio deteriorates, as we show in the next section. Hence, the reduction serves as the main building block for our optimization framework. Subsequently, we describe a polynomial time reduction of the JRC-UR problem to the following variant of the facility location problem [68]:

Definition 3.2. *Unsplittable Hard-Capacitated Metric Facility Location Problem (UHCMFL):* We are given a set $\mathcal{V}$ of $|\mathcal{V}|$ locations, where there is a subset $\mathcal{A} \subseteq \mathcal{V}$ of facilities, and a subset $\mathcal{B} \subseteq \mathcal{V}$ of clients. Let $d_i \geq 0$ denote the demand of client $i \in \mathcal{B}$. Besides, let $f_j \geq 0$ and $C_j \geq 0$ denote the opening cost and the capacity of facility $j \in \mathcal{A}$, respectively. Each client needs to assign its entire demand to a single open facility (*unsplittable*). Capacity C_j limits the total sum of demands served by facility j (*hard capacitated*). We denote by $c_{ij} \geq 0$ the unit cost incurred when serving one unit of demand of client i by facility j. We assume that these costs form a *metric*, i.e., they are non-negative, symmetric ($c_{ij} = c_{ji}$), and satisfy the triangle inequality: $c_{ij} + c_{jk} \geq c_{ik}$, $\forall i, j, k \in \mathcal{V}$.

The problem is to determine which subset of facilities $\mathcal{A}^* \subseteq \mathcal{A}$ should open, and which clients each one of them should serve (denoted by a function $\pi : \mathcal{B} \to \mathcal{A}^*$), so as to minimize the aggregate facility opening and servicing cost Q:

$$Q = \sum_{j \in A^*} f_j + \sum_{i \in \mathcal{B}} d_i c_{i\,\pi(i)}. \tag{3.11}$$

At the same time satisfying the capacity constraints $\sum_{\{i \in \mathcal{B}: \pi(i) = j\}} d_i \leq C_j$, $\forall j \in \mathcal{A}$.

The connection between the UHCMFL and the JRC-UR problem is non-trivial. In fact, previous works in the literature that established reductions of caching problems to facility location problems, focused on the simplified case that only *a single piece of content* is to be placed in the caches [68]. Our model substantially differs from these works, as it considers the practical case that *multiple files* exist, while the cache size and the bandwidth capacity of the SBSs are limited. To the best of our knowledge *this is the first work that shows that such a connection exists.* The following theorem describes this result.

Theorem 3.3. *The JRC-UR problem is polynomial-time reducible to the UHCMFL problem.*

We describe in detail this reduction and prove its validity in the following two subsections.

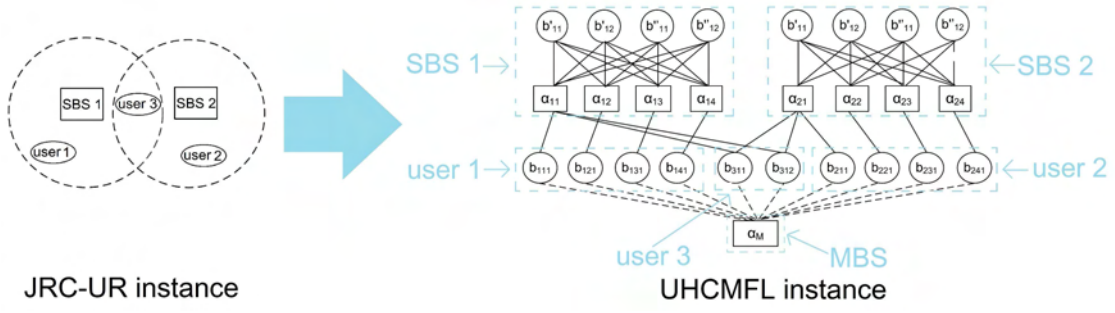


FIGURE 3.3: An example of the reduction to the UHCMFL problem. We consider a setting with 1 MBS, 2 SBSs, and 3 users as shown on the left. The system parameters are $|\mathcal{I}| = 4$, $s = 1$, $S_1 = S_2 = 2$, and $B_1 = B_2 = 2$.

3.3.1 The Reduction

In this subsection, we analytically describe the reduction mentioned in Theorem 3.3. Particularly, we reduce any instance of the JRC-UR problem to an instance of the UHCMFL problem. Let F_{JRC-UR} be that instance of the UHCMFL problem. Then, F_{JRC-UR} is constructed as follows:

The set of facilities $\mathcal{A}$ consists of: (i) one facility named a_M for the MBS and (ii) a facility named a_{ni} for every SBS $n \in \mathcal{N}$ and every file $i \in \mathcal{I}$. The set of clients $\mathcal{B}$ comprises the following subsets: (i) $\mathcal{B}_1$ that contains λ_{ki} clients, $\forall k \in \mathcal{K}$ and $\forall i \in \mathcal{I}$, denoted as b_{ki1}, $b_{ki2} \ldots, b_{ki\lambda_{ki}}$, (ii) $\mathcal{B}_2$, with $|\mathcal{I}| - \lfloor \frac{S_n}{s} \rfloor$ clients, denoted b'_{n1}, b'_{n2} etc, $\forall n \in \mathcal{N}$, and (iii) subset $\mathcal{B}_3$ which contains $(\lfloor \frac{S_n}{s} \rfloor - 1)\lfloor \frac{B_n}{s} \rfloor$ clients, which are denoted b''_{n1}, b''_{n2} etc, $\forall n \in \mathcal{N}$. The symbol $\lfloor . \rfloor$ denotes rounding to the next lower integer. The capacity of the facility a_M is set to $+\infty$ and to B_n/s for each a_{ni}, $\forall n \in \mathcal{N}$, $i \in \mathcal{I}$. The demand of each client $b'_{ni} \in \mathcal{B}_2$ is equal to B_n/s. Each of the remaining clients $b_{kij} \in \mathcal{B}_1$ and $b''_{ni} \in \mathcal{B}_3$ has demand equal to 1.

Let c be an arbitrarily small positive constant. Then, the unit serving cost for each pair of a facility and a client is specified as follows: (i) each pair of the form (a_M, b_{kij}), $\forall k, i, j$, has cost equal to $1+0.5+c$, (ii) each pair of the form (a_{ni}, b_{kij}), such that $n \in \mathcal{N}_k$ and $j \in \{1, ..., \lambda_{ki}\}$, has cost equal to $0.5+c$, (iii) each pair of the form (a_{ni}, b'_{nj}), $\forall n, i, j$, has cost equal to $0.5 + c$, (iv) each pair of the form (a_{ni}, b''_{nj}), $\forall n, i, j$, has cost equal to $0.5 + c$. The cost value of each of the remaining pairs is equal to the cost of the shortest path that unite this pair. Thus, the costs form a metric. Finally, the facility opening cost is set to zero for every facility.

Roughly speaking, the facility a_M represents the MBS and the facilities a_{ni}, $\forall i$, the SBS n. Hence, the facility capacity choices indicate that the MBS can serve all the user requests, while each SBS n can serve up to a limited number of requests. Each one of the clients of the type $b_{kij} \in \mathcal{B}_1$, $\forall k, i, j$ (whose demand is equal to one) represents one

user request, while $b'_{ni} \in \mathcal{B}_2$, and $b''_{ni} \in \mathcal{B}_3$, $\forall n, i$ denote *virtual* user requests that are necessary to preserve the cache capacity and bandwidth constraints of the SBSs, as it will become clear in the following subsection.

Each solution for the F_{JRC-UR} problem can be mapped to a solution for the JRC-UR problem as follows:

- Rule 1: For each facility a_{ni} *not* serving any client of the form $b'_{nj} \in \mathcal{B}_2$, $\forall j$, place file i to the cache of SBS n.
- Rule 2: For each facility of the form a_{ni} serving a client of the form $b_{kij} \in \mathcal{B}_1$, $\forall n, i, k, j$ route the j^{th} request of user k for file i to SBS n.
- Rule 3: The remaining requests are routed to the MBS.

Figure 3.3 depicts the reduction for the example of Figure 3.2. Here, we set the system values as follows: $|\mathcal{I}| = 4$, $s = 1$, $S_1 = S_2 = 2$ and $B_1 = B_2 = 2$. Each of the two first users requests every file once. User 3 performs two requests for the first file. Squares represent the facilities and circles the clients. Solid lines unite clients to facilities with cost $0.5 + c$. Dashed lines mean that the corresponding cost is $1 + 0.5 + c$. The cost value of each of the remaining pairs is equal to the cost of the shortest path that unites this pair. For example, the cost between the client b'_{11} and facility a_{21} is $1.5 + 3c$. The demand of each client is 1, except for the clients named as $b'_{ni} \in \mathcal{B}_2, \forall n, i$, whose demand is 2. The capacity of each facility is 2, except for the a_M facility, whose capacity is $+\infty$.

To help the reader understand the rationality behind the reduction, we also present a partition of the UHCMFL components, specified by the dashed rectangles and the arrowed labels with cyan color. Recall that the role of the clients in $\mathcal{B}_2$ and $\mathcal{B}_3$ is to preserve the cache space and bandwidth limitations of the SBSs, which are mapped to the facilities a_{ni}, $\forall n, i$. Hence, a logical partition of the UHCMFL components should include the above into the same group. To this end, each SBS in our example corresponds to the 8 components in the top of the UHCMFL instance (namely 4 facilities and 4 clients), each user k to $\sum_{i \in \mathcal{I}} \lambda_{ki}$ of the bottom clients, and the MBS to the facility named a_M.

3.3.2 The Reduction Proof

We now prove that the preceding reduction holds, by proving the next two lemmas. Let D denote the total demand of the clients in F_{JRC-UR}. Then, we have:

Lemma 3.4. *For every feasible solution of the JRC-UR problem with value C, there is a feasible solution to F_{JRC-UR} with total cost $C + D(c + 0.5)$.*

We construct the solution to F_{JRC-UR} as follows:

1. We open all the facilities at zero cost.
2. For each file i *not* cached at SBS n, we assign the (entire) demand of one client of type $b'_{nj} \in \mathcal{B}_2$, $j \in \{1, ..., |\mathcal{I}| - \lfloor \frac{S_n}{s} \rfloor\}$ to the facility a_{ni}.
3. For each request generated by a user k for a file i served by an SBS n, we assign the demand of one client of type $b_{kij} \in \mathcal{B}_1$, $j \in \{1, ..., \lambda_{ki}\}$ to the facility a_{ni}.
4. The demand of a client of type $b''_{nj} \in \mathcal{B}_3$, $j \in \{1, ..., (\lfloor \frac{S_n}{s} \rfloor - 1)(\lfloor \frac{B_n}{s} \rfloor)\}$, is randomly assigned to one of the facilities of the form a_{ni}, $\forall i \in \mathcal{I}$, without violating their capacity constraints.
5. For each client $b_{kij} \in \mathcal{B}_1$ that has not been covered yet, we assign its demand to the facility a_M. Thus, every unit of demand of the clients was assigned to a facility. An assignment to the facility a_M incurs a per unit cost equal to $1 + 0.5 + c$, while all the other assignments incur a per unit cost equal to $0.5 + c$. By construction of the graph, the total demand assigned to a_M is equal to the number of requests that are routed to the MBS (C). Thus, the solution has cost equal to $C + D(0.5 + c)$.

Lemma 3.5. *For every minimum cost solution of the F_{JRC-UR} instance with total cost C, there is a feasible solution to the JRC-UR problem with value $C - D(0.5 + c)$.*

We construct the solution to JRC-UR problem as follows:

(i) *For each facility a_{ni}* not *serving any client of the form $b'_{nj} \in \mathcal{B}_2$, $\forall j$, place file i to the cache of SBS n* (Rule 1). Observe that each client $b'_{nj} \in \mathcal{B}_2$, $\forall j$, must be assigned to a facility of the form a_{ni}, $\forall i \in \mathcal{I}$, at per unit cost $0.5 + c$. This is because each of the other choices incurs at least $1 + 0.5 + 3c$ per unit cost. Thus, the extra cost paid is at least $1 + 2c$. On the other hand, each client $b_{kij} \in \mathcal{B}_1$, $\forall k, i, j$, can always be assigned to the facility a_M at per unit cost $1 + 0.5 + c$. This means that the potential gain for assigning it to a facility of the form a_{ni}, $\forall i \in \mathcal{I}$, at cost $0.5 + c$, is equal to 1, which is strictly lower than the extra cost paid above. We also observe that, the demand of each of these clients is equal to the capacity of each of the facilities of the form a_{ni}, $\forall i \in \mathcal{I}$. There are $|\mathcal{I}| - \lfloor \frac{S_n}{s} \rfloor$ such clients. Thus, these clients fully occupy the capacity of $|\mathcal{I}| - \lfloor \frac{S_n}{s} \rfloor$ of these facilities. Consequently, exactly $\lfloor \frac{S_n}{s} \rfloor$ of the above facilities will remain *uncovered* corresponding to the files placed at the cache of SBS n.

(ii) *For each facility of the form* a_{ni} *serving a client of the form* $b_{kij} \in \mathcal{B}_1$, $\forall n, i, k, j$ *route the* j^{th} *request of user* k *for file* i *to SBS* n (Rule 2). Observe that each of the clients of type $b''_{nj} \in \mathcal{B}_3$, $\forall j$, must be assigned to one of the $\lfloor \frac{S_n}{s} \rfloor$ *uncovered* facilities of the form a_{ni}, $\forall i$, similarly to the above case. The capacity of each of these facilities is equal to $\frac{B_n}{s}$. There exist $(\lfloor \frac{S_n}{s} \rfloor - 1)\lfloor \frac{B_n}{s} \rfloor$ such clients, each of them with demand equal to 1. Thus, the remaining capacity suffices for serving at most $\frac{B_n}{s}$ units of demand of the clients $b_{kij} \in \mathcal{B}_1$, $\forall k, i, j$. By construction, a client $b_{kij} \in \mathcal{B}_1$ can be served by a facility a_{ni} with cost equal to $0.5 + c$ iff $n \in \mathcal{N}_k$. The cost for serving $b_{kij} \in \mathcal{B}_1$ by a_{ni}, $\forall n \notin \mathcal{N}_k$ is more than the serving cost by a_M. Thus, at most $\frac{B_n}{s}$ requests generated by users in the coverage area of an SBS n will be routed to $n, \forall n \in \mathcal{N}$. The remaining $C - D(0.5 + c)$ requests will be routed to the MBS (Rule 3).

To avoid confusion, we need to emphasize that the key point of the reduction is to force all the clients $b'_{nj} \in \mathcal{B}_2$ and $b''_{nj} \in \mathcal{B}_3$, $\forall j$ to assign their (entire) demand on one of the facilities of the form a_{ni}, $\forall i$, for each SBS $n \in \mathcal{N}$. To ensure this, we picked the servicing cost values appropriately. A different choice would be to set the cost between each one of the clients in the set $\mathcal{B}_2 \cup \mathcal{B}_3$ and one of the above facilities equal to zero, and to $+\infty$ for all the other choices. Then, we could set the cost for assigning clients in $\mathcal{B}_1$ from $0.5 + c$ to zero, from $1 + 0.5 + c$ to 1 and to $+\infty$ for the rest choices. Note that, although this new instance of the facility location problem would be equivalent to the JRC-UR problem, the costs would not form a metric any more. The non-metric version of the UHCMFL problem is much harder, and there are not any approximation algorithms for it in the literature. This is the reason that we added the quantity $0.5 + c$ to the cost value of each one of the above links and restricted the cost of each one of the rest links to be equal to the aggregate cost in the shortest path that unites the endpoint vertices.

Note that the reduction does not hold for $c \leq 0$, since the extra cost paid for assigning a client $b'_{nj} \in \mathcal{B}_2$ or $b''_{nj} \in \mathcal{B}_3$ to a facility $a_{n'i}$, $n' \neq n$ can be lower than the potential gains.

3.4 Approximation Algorithms

In this section, we present an approximation framework for the JRC-UR problem based on the reduction described in Sec. 3.3. We first discuss existing approximation algorithms for the UHCMFL problem. Accordingly, we describe methods to extend them so as to tackle the problem under consideration. Additionally, we derive improved approximation ratios for the special case of equal transmission capacity SBSs. This is an important case because, more often than not, SBSs will be of the same type and hence they will have equal transmission capacities.

3.4.1 Approximation Ratios for the UHCMFL Problem

It is NP-hard even to approximate the solution of UHCMFL problem. Hence, previous works [64]-[69] focused on obtaining *bi-criteria* approximation algorithms. Formally, an $(\alpha, \beta)-$bi-criteria approximation algorithm finds an infeasible solution with a cost at most $\alpha \geq 1$ times the optimal cost and aggregate demand assigned to each facility at most $\beta \geq 1$ times its capacity. Similarly, we can define an $(\alpha, \beta)-$bi-criteria approximation algorithm for the JRC-UR problem, such that its solution violates the bandwidth capacities of the SBSs by at most a factor of β. Clearly, when β equals to one, a feasible solution is attained. We call the corresponding algorithm simply as an $\alpha-$approximation algorithm.

Table 3.1 summarizes the existing results in the literature in chronological order. Notice that different results are obtained for the case that facilities have equal capacities (uniform case). Parameter $\epsilon > 0$ is arbitrarily small and $|\mathcal{V}|$ is the size of the facility location instance. A Quasi-polynomial time algorithm ($QP.$) runs slower than polynomial time ($P.$), yet faster than exponential time [68]. For example, the complexity of Algorithm 3.7 is $|\mathcal{V}|^{O(\log |\mathcal{V}|)}$.

Shmoys et al. [64] provided the first approximation algorithm for the UHCMFL problem (Algorithm 3.1). They used the filtering and rounding technique of Lin and Vitter to solve the splittable version of the problem, and then round the obtained solution to provide a $(9, 4)$-approximation algorithm for the unsplittable case. The first step requires solving the linear programming relaxation of the UHCMFL problem, and then rounding it to obtain a $g - close$ integer solution, i.e., a solution that assigns clients to facilities with cost at most g. The second step expresses the problem as one of assigning jobs to machines. Then, it rounds the current solution to a new one by solving an appropriately constructed instance of the maximum weight matching in a bipartite graph problem. The running time of this procedure is cubic to the size of the facility location instance. Authors in [64] also proposed a randomized variant of the above technique that provides an improved approximation guarantee (Algorithm 3.2). As it is well discussed in [69], applying the same rounding technique of [64] to the results for the splittable case provided in [65], [66], [67] yields even tighter approximation ratios for the unsplittable variant of the facility location problem (Algorithms 3.3-3.5).

The work in [68] provided an $(1 + \epsilon, 1 + \epsilon)-$approximation algorithm for the UHCMFL problem for the special case that the costs form a tree metric [71], $\forall \epsilon > 0$. Their algorithm is based on a dynamic programming approach. Clearly, the costs in the instance of the facility location problem in Section 3.3 do not form a tree metric. Hence, we can not use this result as the previous ones. Interestingly, using Fakcharoenphol

TABLE 3.1: Bi-criteria Bounds for the UHCMFL problem.

Algorithm	Case	Bound	Complexity	Reference
3.1	Uniform	$(9, 4)$	P.	[64]
3.2	Uniform	$(7.62, 4.29)$	P.	[64]
3.3	Uniform	$(O(1), 2)$	P.	[65]
3.4	General	$(11, 2)$	P.	[69],[66],[64]
3.5	Uniform	$(5, 2)$	P.	[69],[67],[64]
3.6	Uniform	$(\log \lvert\mathcal{V}\rvert, 1+\epsilon)$	P.	[68]
3.7	General	$(\log \lvert\mathcal{V}\rvert, 1+\epsilon)$	QP.	[68]
3.8	Uniform	$(10.173, 1.5)$	P.	[69]
3.9	Uniform	$(30.432, \frac{4}{3})$	P.	[69]

et al.'s machinery [71], we can translate the above solution to obtain a $(\log |\mathcal{V}|, 1+\epsilon)$-approximation algorithm for general metrics. This translation requires polynomial time for the uniform capacities case and quasi-polynomial time for the general case. Finally, the work in [69] provided the first approximation algorithms that violate the capacities by a factor less than two and achieve a constant approximation ratio. The result is based on a reduction to a restricted version of the initial problem in a way that any $(O(1), 1+\epsilon)$-approximation algorithm for the restricted problem implies an $(O(1), 1+\epsilon)$-approximation algorithm for the initial problem, for $\epsilon \in \{1/2, 1/3\}$.

3.4.2 Approximation Ratios for the JRC-UR Problem

Although JRC-UR and UHCMFL problems are equivalent in terms of their optimal solution, the extension of approximation algorithms from one to the other is not straightforward. The following theorem describes the way that the bi-criteria bound changes when translating the solution to handle the JRC-UR case. Let us define:

$$c' = \frac{D(0.5 + c)}{\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} (\lambda_{ki}) - \sum_{n\in\mathcal{N}} (B_n/s)} \tag{3.12}$$

Then, we have:

Theorem 3.6. *For any $(\alpha, \beta)-$bi-criteria approximation algorithm for the UHCMFL problem there is an $\big(\alpha + (\alpha-1)c', (\beta-1)|\mathcal{I}| + 1\big)-$bi-criteria approximation algorithm for the JRC-UR problem, requiring the same computational complexity.*

The overall traffic routed to an SBS $n \in \mathcal{N}$ by the real users is $\beta|\mathcal{I}|\frac{B_n}{s} - (|\mathcal{I}| - \lfloor\frac{S_n}{s}\rfloor)\frac{B_n}{s} - (\lfloor\frac{S_n}{s}\rfloor - 1)\lfloor\frac{B_n}{s}\rfloor$ in the worst case. This is because, each SBS corresponds to $|\mathcal{I}|$ facilities, each one of which has capacity equal to the capacity of the SBS. The virtual clients of the type $b'_{nj} \in \mathcal{B}_2$ and $b''_{nj} \in \mathcal{B}_3$, $\forall j$, must be served by the facilities a_{ni} $\forall i$, in any case,

as explained in the previous section. That is the reason that we subtracted the traffic sent to them in the above expression. However, in reality, only $\frac{B_n}{s}$ amount of data can be transmitted by each SBS. The fraction of the two values, after some computations, can be written as $(\beta - 1)|\mathcal{I}| + 1$.

Besides, let *opt* and *approx* be the optimal and an approximation solution of the JRC-UR problem respectively. By Lemma 3.4, it holds that:

$$approx + D(0.5 + c) \leq \alpha\big(opt + D(0.5 + c)\big) \tag{3.13}$$

or equivalently:

$$approx \leq \big(\alpha + \frac{D(0.5 + c)(\alpha - 1)}{opt}\big)opt \tag{3.14}$$

Finally, it is:

$$opt \geq \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} (\lambda_{ki}) - \sum_{n \in \mathcal{N}} (B_n/s) \tag{3.15}$$

which completes the proof.

Theorem 3.6 combined with Algorithms 3.4 and 3.7 in Table 3.1, provides two bi-criteria approximation algorithms for the JRC-UR problem. The following corollary describes this result:

Corollary 3.7. *There exist a* polynomial time *(α_1, β_1)-bi-criteria approximation algorithm and a* quasi-polynomial time *(α_2, β_2)-bi-criteria approximation algorithm for the JRC-UR problem, for:*

$$(\alpha_1, \beta_1) = (11 + 10c', |\mathcal{I}| + 1) \tag{3.16}$$

$$(\alpha_2, \beta_2) = (\log v + (\log v - 1)c', \epsilon|\mathcal{I}| + 1), \epsilon > 0 \tag{3.17}$$

where v is the size of the instance of the UHCMFL problem corresponding to the JRC-UR problem.

We can use the above bi-criteria solutions to perform caching and routing in our problem as it is described in Figure 3.4. Note also that, as the bandwidth capacities of the SBSs may be violated by the above factors, the operator may need to endow the base stations with additional bandwidth capacity, in order to ensure the described approximation ratio. Nevertheless, in many cases, the operator is unwilling (or, incapable) to perform additional investments. Thus, the additional requests that reach an SBS will be rerouted to the MBS, further increasing its load. How much worse is the obtained result? The next theorem characterizes the worst case scenario in terms of the quality of the resulted solution.

FIGURE 3.4: Our algorithms operate in three stages: In Stage I, the MNO transforms the JRC-UR into the UHCMFL instance. In Stage II, it solves that instance by employing one of the algorithms in Table 3.1. In Stage III, it maps the obtained solution to a solution for the JRC-UR instance based on the rules in Section 3.3

Theorem 3.8. *For any $(\alpha, \beta)-$bi-criteria approximation algorithm for the UHCMFL problem there is an $(\alpha+(\alpha-1)c')c''(\beta)-$approximation algorithm for the JRC-UR problem, requiring the same computational complexity, where:*

$$c''(\beta) = \frac{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}(\lambda_{ki}) - \sum_{n\in\mathcal{N}}(B_n/s)}{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}(\lambda_{ki}) - ((\beta-1)|\mathcal{I}|+1)\sum_{n\in\mathcal{N}}(B_n/s)} \tag{3.18}$$

Let H_β be the number of requests routed to the SBSs and R_β be the number of requests routed to the MBS, according to the described reduction, when the capacities of the facilities are violated by a factor of β. In reality, all the requests beyond the capacities of the SBSs will be rerouted to the MBS, as the SBSs can not serve them. Let H be the number of requests served by the SBSs and R the number of requests served by the MBS after this rerouting. Then, it holds:

$$\begin{aligned}
\frac{R}{R_\beta} &= \frac{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}(\lambda_{ki}) - H}{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}(\lambda_{ki}) - H_\beta} \\
&= \frac{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}(\lambda_{ki}) - H}{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}(\lambda_{ki}) - H\cdot((\beta-1)|\mathcal{I}|+1)} \\
&= \frac{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}(\lambda_{ki}) - \sum_{n\in\mathcal{N}}\lfloor\frac{B_n}{s}\rfloor}{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}(\lambda_{ki}) - \sum_{n\in\mathcal{N}}\lfloor\frac{B_n}{s}\rfloor\cdot((\beta-1)|\mathcal{I}|+1)}
\end{aligned} \tag{3.19}$$

where the first equation holds because of the definition of the terms H, H_β, R and R_β which yields that: $H_\beta + R_\beta = H + R = \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}(\lambda_{ki})$. The second equation is because of theorem 3.6: $H_\beta = H((\beta-1)|\mathcal{I}|+1)$. Finally, the last equation holds because after the rerouting of requests each SBS will only serve as many requests as its capacity allows.

Theorem 3.8 combined with the results in Table 3.1, provides two approximation algorithms for the JRC-UR problem:

Corollary 3.9. *There exists a* polynomial time *α_3-approximation algorithm and a* quasi-polynomial time *α_4-approximation algorithm, for the JRC-UR problem, for:*

$$\alpha_3 = (11+10c')c''(2) \tag{3.20}$$

$$\alpha_4 = (\log v + (\log v - 1)c')c''(1+\epsilon), \epsilon > 0, \tag{3.21}$$

where v is the size of the instance of the UHCMFL problem corresponding to the JRC-UR problem.

3.4.3 The Case of Uniform-capacity SBSs

In this subsection, we focus on the special case of the JRC-UR problem where the SBSs have equal transmission capacities, i.e., $B_n = B, \forall n \in \mathcal{N}$. For example, assume that SBSs are of the same type, e.g., certain type of femtocells or picocells. However, the cache sizes can be different. We can map this problem to a certain UHCMFL problem in which all the capacities of the facilities are *equal* and exploit the improved approximation ratios that are known for this uniform capacity setting. According to the reduction described in Section 3.3.1, for this special case of the problem, all the capacities of the facilities are equal, i.e., $C_j = B/s, \forall j \in \mathcal{A}$, except for the facility a_M, which has infinite capacity. Parameter a_M can be replaced by $\lceil \frac{\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \lambda_{ki}}{\lfloor B/s \rfloor} \rceil$ facilities each one of capacity B/s. Clearly, the aggregate capacity of them suffices to serve all the demand of the clients of the form $b_{kij} \in \mathcal{B}_1$, $\forall k, i, j$ and the new instance is equivalent to the initial.

Based on the above, Table 3.1 provides the following approximation algorithms for the uniform-capacity JRC-UR problem. The next corollary describes the results:

Corollary 3.10. *For the uniform-capacities JRC-UR problem, there exists an r_1-bi-criteria approximation algorithm, and an r_2-approximation algorithm, such that:*

$$r_1 = \big(\alpha + (\alpha - 1)c', (\beta - 1)|\mathcal{I}| + 1\big) \tag{3.22}$$

$$r_2 = \big(\alpha + (\alpha - 1)c'\big)c''(\beta) \tag{3.23}$$

$$\textit{for: } (\alpha, \beta) \in \{(9, 4), (7.62, 4.29), (O(1), 2), (5, 2), (\log v, 1 + \epsilon),$$
$$(10.173, 1.5), (30.432, 4/3)\}, \epsilon > 0,$$

where v is the size of the instance of the UHCMFL problem corresponding to the JRC-UR problem.

3.5 Performance Evaluation

In this section, we present the numerical results of the experiments that we have conducted to evaluate our derived theoretical results. Specifically, using realistic system settings we characterize the performance improvements offered by one of the proposed algorithms over conventional caching schemes, as well as its performance gap to the optimal solution.

3.5.1 Simulation Setup and Methodology

We compare the performance of the four following schemes:

1. *Greedy*: The naive approach according to which each SBS caches the most popular files based on the requests of the nearby users independently from the others. When a request is generated, it is routed to the nearest SBS that has stored a copy of the associated file.

2. *Iterative* [33]: It starts with all the caches empty. At each iteration, it places the file to a non-full cache that yields the lowest value of the objective function in (3). The algorithm terminates when all the caches become full. When a request is generated, it is routed to the nearest SBS that has stored a copy of the associated file.

3. *Facility*: The routing and caching policies are jointly derived by solving the instance of the facility location problem using Algorithm 3.1 in Table 3.1, as described in detail in Section 3.4, and it is shown in Figure 3.4.

4. *Optimal*: The optimal solution of the JRC-UR problem found through exhaustive search. Since its running time is unacceptable large, i.e., in the scale of days, using realistic system settings, Optimal is only used as a benchmark for gauging the performance of the proposed solutions and determine if there is still room for improvement.

The performance criterion we use is the total number of requests that reach the MBS (*MBS load*). To describe in detail the performance improvements of the Facility scheme compared to its alternatives, we also depict the normalized difference between the MBS load achieved by any of the first three schemes and the Optimal (*MBS load difference*). Formally, the MBS load difference of the Greedy algorithm is defined as:

$$\frac{MBS\_load_{Greedy} - MBS\_load_{Optimal}}{MBS\_load_{Optimal}} \tag{3.24}$$

where MBS_load_{scheme} denotes the MBS load achieved by the associated scheme, i.e., the value $\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} y^k_{Mi}$. A similar definition holds for the Iterative and the Facility Scheme.

Throughout, we consider a single MBS serving a circular-shaped cell with radius 350 meters (typical of urban macrocell [33]). We assume that $N = 16$ SBSs are randomly deployed within it, each one having a communication range of 80 meters. The system supports the service of a rich collection of $I = 1,000$ unit-sized files. Unless otherwise

specified, a large number of $K = 1,000$ mobile users are uniformly placed in random statistically independent positions in the cell. Each user requests one file based on the zipf law with shape parameter $z = 0.8$ [61], i.e., the probability that a request is for the j^{th} most popular file is: $j^{-z}/\sum_{i=1}^{|\mathcal{I}|} i^{-z}$. Each SBS n is endowed with a cache of size $S_n = S$, $\forall n \in \mathcal{N}$ that is equal to 3% of the entire file set size. Finally, its bandwidth $B_n = B$, $\forall n \in \mathcal{N}$ suffices for transmitting 5% of the entire file set.

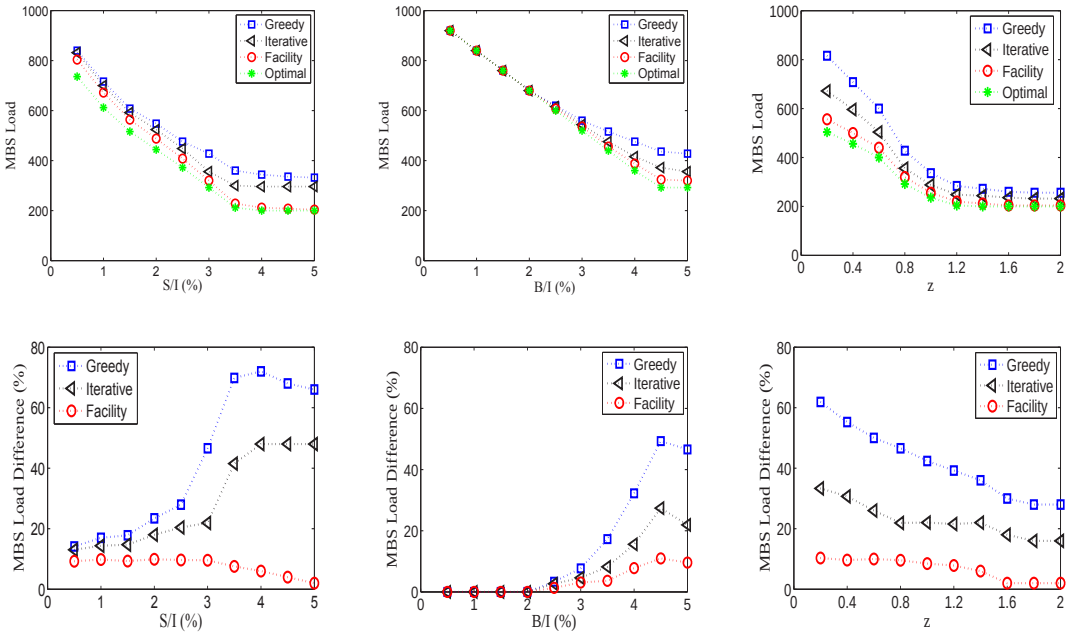
3.5.2 Parameter Impact Analysis

3.5.2.1 Impact of the Cache Sizes

Figures 3.5(a) compare the performance of the discussed schemes as a function of the cache size S of each SBS. Parameter S varies in our simulation from 0.5% to 5% of the entire file set size. As expected, increasing the available cache space, decreases the MBS load for all the schemes, as more files are cached at the SBSs. More importantly, as S increases the performance of Facility scheme comes very close to the optimal one. Even for low values of S, the Facility scheme operates very close to the Optimal (less than 10% worse), and far better than the worst case conditions indicate. Besides, the Facility scheme provides significant performance gains, up to 38%, over the Greedy and Iterative schemes. To elaborate on this, we observe that although the Iterative scheme performs the cache placement more efficiently than Greedy (since it places the files in multiple stages rather than simultaneously), both schemes fail to appropriately route the user requests to the SBSs. This is because, they both ignore the bandwidth limitations of the SBSs (bandwidth-agnostic).

3.5.2.2 Impact of the Transmission Bandwidth Capacities

We analyze the impact of the transmission bandwidth capacities on the algorithms' performance in Figures 3.5(b). We vary the bandwidth capacity per SBS B from 0.5% to 5% of the entire file set size. As expected, increasing B, decreases the MBS load, since the SBSs can serve more requests. We observe that for low values of B, i.e., when the system is in overloaded conditions, the performance of the three schemes is similar. This is because, in these cases, simply caching the S most popular file at each SBS suffices to fully utilize its bandwidth capacity for almost all the SBSs. Interestingly, the performance gap between Facility and Optimal scheme increases as B increases in the range of 0 to 10%. This is because, as explained in Section 3.4, the solution of the facility location problem may violate the bandwidth capacities of the SBSs, and redirecting the extra requests to the MBS further increases its load. This is more crucial for high values



(a) Cache Size (b) Bandwidth Capacity (c) Zipf-Parameter

FIGURE 3.5: Performance comparison between Greedy, Iterative, Facility and Optimal scheme for various values of (a) the cache size, (b) the bandwidth capacity per SBS and (c) the zipf parameter of the popularity distribution of the files.

of B. Finally, we note that Facility scheme consistently outperforms the Greedy and Iterative schemes, a gap that increases with B in the range of 0% to 25%.

3.5.2.3 Impact of the File Request Pattern

We explore the impact of the steepness of the file request pattern on the algorithms' performance in Figures 3.5(c). Namely, we vary the shape parameter z of the file popularity from the value 0.2 to 2. We observe that as z increases, the MBS load decreases for all the schemes, reflecting the well known fact that caching effectiveness improves as the popularity distribution gets steeper. Besides, as z increases the performance gap between each pair of the discussed algorithms is shrinking. This is because, when z is high, the vast majority of user requests refer to a small number of files. Clearly, caching the above files provides significant benefits to the provider. To conclude, our algorithm achieves a performance that is up to 31% better than the Greedy, up to 17% better than Iterative, and less than 10% worse than the optimal one.

Access	Speed(Mbps)	Monthly price(euros)	%residential users
ADSL	24/1	21.94	28.7
Fiber	30/3	25.20	6.6
Fiber	100/10	33.33	64.7

TABLE 3.2: Tariff plans for all the residential users in our dataset.

3.6 Extension to the Case of Residential User-owned Caches

In this section, we extend our model for the case that the caches are owned by residential WiFi users. Considering that the SBS deployment requires significant CAPEX cost to MNO, this architecture can significantly reduce costs. In fact, residential users today own possibly more than one storage devices (e.g., hard disks, flash memories, etc.), which can be used by the MNO to cache and deliver content to mobile users. The success of this proposal depends on the willingness of the residential users to lease their cache space and wireless bandwidth. Intuitively, residential users that frequently connect to the Internet through their wireless devices (e.g., laptops, tablets, smartphones) are expected to be more reluctant for leasing the wireless bandwidth of their Access Points (APs). The MNO can offer monetary incentives (reimbursements) to compensate for the opportunity cost, i.e., the fact that due to the reduced bandwidth, the residential users may need to change their daily habits, such as the time of the day they use their devices, the type of applications they run, etc. The required reimbursement is expected to be higher when the utilization of the WiFi connection inside residence increases.

3.6.1 Dataset Analysis

Clearly, it is important to understand *how often* and *when* the residential WiFi APs are utilized. In order to obtain insights into this question, we analyze a dataset of detailed WiFi usage obtained from 167 residential users in Portugal, subscribers of Portugal Telecom (PT). These users are distributed over a large geographic area spanning 10 cities, and volunteered to be part of a data collection project within a 4-month period (June-September, 2013). The gateway platform of each user has the following specifications: (a) ADSL2+ modem or fiber WAN access link, (b) 4 ethernet ports and (c) a WiFi AP with a Broadcom 802.11b/g/n 2x2 interface with MIMO support. The 802.11 interface operates at 2.4GHz band and supports both 20MHz and 40MHz channels. The tariff plans span two different access technologies (ADSL and Fiber) and three different downstream/upstream access speeds at the backhaul links (24/1, 30/3 and 100/10 Mbps). Table 3.2 presents a summary of the subscriber population in our dataset.

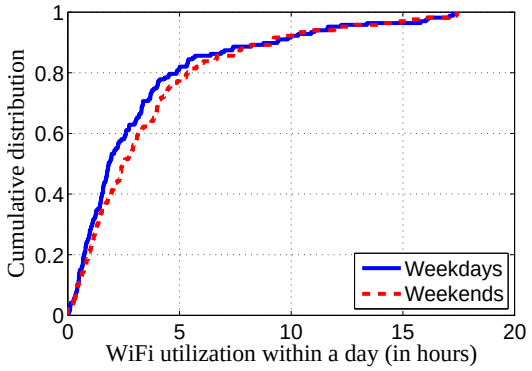


FIGURE 3.6: Cumulative Distribution Function of WiFi utilization within a day.

Each gateway reports every 30 seconds the *actual throughput* T_a (measured in Kbps) for each wireless device connected to it. The actual throughput is the total number of (transmitted or received) bits over time, and it captures the actual demand on the wireless network. We leverage T_a metric to estimate *WiFi utilization* over time. Particularly, we consider a WiFi AP to be utilized across a 30-second time bin t, if there is at least one in-residence device with non-zero aggregated (transmit and receive) actual throughput during t. During the utilized periods, serving external (non-resident) devices may negatively impact the residential users' quality of experience.

In Figure 3.6, we plot the cumulative distribution function of WiFi utilization within a day (measured in hours) seen across the 167 users in our dataset. Here, measurements for weekdays and weekend days are taken separately. We find that WiFi utilization is slightly higher during weekend days than weekdays, with the average values being 3.69 and 3.32 hours per day respectively. We also observe a *wide disparity in WiFi utilization* in both weekdays and weekend days. The majority of the users use WiFi for a few hours within a day, with $\sim 80\%$ of them being active for less than 5 hours of the day. On the upper extreme, a few users connect to WiFi for up to 18 hours of the day. This is an important result, since the unused wireless bandwidth can be leased by the MNO without actually affecting the quality of experience of the residential users. To reduce its leasing fees, the MNO should target the residential users with the lowest WiFi utilization among a set of candidate choices in the same region.

In order to capture the temporal characteristics of WiFi utilization, we depict in Figure 3.7 the probability that the APs are utilized during different times of the day. For a specific time, this probability is computed by summing all the WiFi measurements collected in our dataset for this time, and normalizing by the number of samples. We find that WiFi utilization follows a diurnal pattern, with the value increasing during the daytime, but reducing at late night until the early morning. The peak value (found

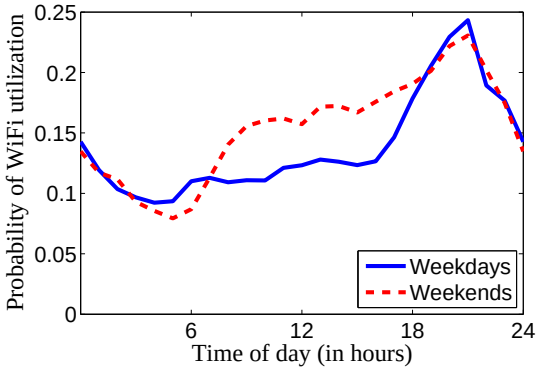


FIGURE 3.7: Daily pattern of WiFi utilization.

to be at 21pm in the evening) is ~ 2.5 times higher than the lowest one (found to be at 5am in the early morning) during both weekdays and weekend days. During the weekend, the WiFi utilization is higher for a longer period of time, reflecting the users accessing the Internet from home during the morning too. Interestingly, *most of the wireless bandwidth is unused during all times of the day, even in the evening.* This is of high importance, since an MNO may need to offload the cellular network in the evening when the mobile data traffic may also peak.

Summarizing, in circumstances that cover most kinds of residential user activity, leasing WiFi bandwidth can be a feasible mechanism for the MNO to reduce the mobile data volumes. However, the overall cost-benefit is likely to depend on a number of factors: response of the residential users to incentives, network costs of MNO, spatiotemporal characteristics of mobile user demand, and so on.

3.6.2 Residential User Model

In the new model, we use the set $\mathcal{N}$ to denote N residential users rather than SBSs. Each residential user owns a WiFi access point (AP) with transmission range a few tens of meters, e.g., using IEEE 802.11b/g/n technology. We assume that the residential users are rational and self-interested entities. Namely, we denote with $U_n^B(r)$ the utility perceived by residential user n for using r portion of his wireless bandwidth to serve his own communication needs (i.e., to serve the registered devices in the residence). Similarly, we denote with $U_n^S(r)$ the obtained utility for using r portion of his cache capacity to cache data for his own needs. The utility functions may change over time (e.g., due to variations in habits and behavior of the users), but they are considered fixed for the period of study. Following the principle of diminishing returns [72], we consider residential user utility to be logarithmically related to the portion of consumed

resources:

$$U_n^B(r) = \alpha_n^B \log(\beta_n^B \cdot r) \tag{3.25}$$

and

$$U_n^S(r) = \alpha_n^S \log(\beta_n^S \cdot r) \tag{3.26}$$

Here, α_n^B is a scaling parameter that captures how different users evaluate a unit of wireless bandwidth. Parameter β_n^B specifies the slope of the logarithmic function for the bandwidth. Large β_n^B values reflect an indifference to bandwidth variations, while low values reflect that the user is affected by changes in bandwidth. Similarly, parameters α_n^S and β_n^S characterize the residential user valuation for the cache space.

The AP of a residential user n can cache and deliver content to the nearby mobile users. This induces cost to the residential user due to energy consumption. Following previous work, we consider the caching cost to increase as more data is cached, controlled by a slope parameter π_n^S, i.e., it will be equal to $\pi_n^S \cdot b$ when b portion of the cache capacity is leased [73]. Similarly, the cost for delivering mobile data will be $\pi_n^B \cdot b$ when b portion of the wireless bandwidth capacity is leased [99].

3.6.3 MNO Model

The MNO can lease cache and wireless bandwidth capacity from the residential users to offload MBS. To accomplish this, the MNO announces to residential users that they will benefit from price discounts that are proportionally related to the amount of leased resources. Particularly, the MNO announces a price $p_n^S \geq 0$ per portion of cache space and a price $p_n^B \geq 0$ per portion of wireless bandwidth capacity that will offer to each residential user $n \in \mathcal{N}$ as a leasing fee. We emphasize that the prices may differ across residential users for two reasons. First, the MNO may assume that some residential users will be more or less reluctant to accept mobile data traffic and cache content. Second, the cost savings from offloading may be higher or lower for different APs depending on their location, e.g., at the cell edge the energy savings from offloading may be higher compared to a location close to MBS. The respective vectors can be defined as follows:

$$p^S = (p_n^S \ : \ n \in \mathcal{N}), \ p^B = (p_n^B \ : \ n \in \mathcal{N}) \tag{3.27}$$

In response to these offers, the MNO leases $\widehat{S_n}(p_n^S) \in [0,1]$ portion of cache space and $\widehat{B_n}(p_n^B) \in [0,1]$ portion of WiFi bandwidth from residential user n. Since the residential users are rational and self-interested, it is expected to determine these quantities by assessing the benefits from using the AP resources for their own needs and the benefits from the offered monetary incentives. In particular, the portion of leased cache space

for residential user n will be:

$$\widehat{S_n}(p_n^S) = \underset{b\in[0,1]}{\text{argmax}}[U_n^S(1-b) + p_n^S \cdot b - \pi_n^S \cdot b] \tag{3.28}$$

i.e., he will decide how much cache space he will lease by maximizing the sum of utility and payment obtained minus the cost for caching content. Similarly, the portion of leased wireless bandwidth will be:

$$\widehat{B_n}(p_n^B) = \underset{b\in[0,1]}{\text{argmax}}[U_n^B(1-b) + p_n^B \cdot b - \pi_n^B \cdot b] \tag{3.29}$$

Given the utility function shapes in equations (3.25)-(3.26), the leased portions for each residential user n are given by [82]:

$$\widehat{S_n}(p_n^S) = 1 - \frac{\alpha_n^S}{p_n^S - \pi_n^S},\ \widehat{B_n}(p_n^B) = 1 - \frac{\alpha_n^B}{p_n^B - \pi_n^B} \tag{3.30}$$

where, in order to be $\widehat{S_n}(p_n^S) \in [0,1]$ and $\widehat{B_n}(p_n^B) \in [0,1]$, it should hold $p_n^S - \pi_n^S \geq \alpha_n^S$ and $p_n^B - \pi_n^B \geq \alpha_n^B$.

The MNO can predict the response of the residential users to any possible prices in $p^S,\ p^B$. In other words, the MNO can learn the functions $\widehat{B_n}(p_n^B)$ and $\widehat{S_n}(p_n^S)$, $\forall n \in \mathcal{N}$. Even if the price vectors (*incentive policy*) are decided, the MNO needs to additionally decide which content files to place in each cache (*caching policy*) and in which AP to route the requests of each mobile user (*routing policy*). These are constrained by the amount of cache and bandwidth capacity leased. Hence, equations 3.4 and 3.5 should be replaced by the following:

$$\sum_{i\in\mathcal{I}} x_{ni} s \leq \widehat{S_n}(p_n^S) \cdot S_n,\ \forall n \in \mathcal{N}, \tag{3.31}$$

$$\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} y_{ni}^k s \leq \widehat{B_n}(p_n^B) \cdot B_n,\ \forall n \in \mathcal{N}, \tag{3.32}$$

where B_n (bytes per second) indicates the expected bandwidth rate with which the AP n can transmit data to mobile users. This is expected to be lower than the nominal access rate of the AP router due to interference from coexisting WiFi-transmitters and commonly available non-WiFi devices such as mircrowave ovens, cordless phones, etc [102], [101].

Also, the MNO should target to minimize the total cost for both serving the mobile users and providing incentives to residential users:

$$\min_{x,y,p^S,p^B} Q(y) + \sum_{n\in\mathcal{N}} \left(p_n^S \widehat{S_n}(p_n^S) + p_n^B \widehat{B_n}(p_n^B) \right) \tag{3.33}$$

where $Q(y)$ denotes the cost for serving requests by the MBS. In our recent work in [22], we used the primal-dual method to solve the joint caching, routing and incentive problem. Specifically, we relaxed the constraint in (3.6) and introduced the set of dual Lagrange multipliers:

$$\mu = (\mu_{ni}^k \geq 0 : \forall\, n \in \mathcal{N}, i \in \mathcal{I}, k \in \mathcal{K}) \tag{3.34}$$

Then, we defined the Lagrange function as follows:

$$\begin{aligned} L(x, y, p^S, p^B, \mu) = Q(y) \\ + \sum_{n\in\mathcal{N}} \left(p_n^S \widehat{S_n}(p_n^S) + p_n^B \widehat{B_n}(p_n^B) \right) \\ + \sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \mu_{ni}^k (y_{ni}^k - x_{ni}) \end{aligned} \tag{3.35}$$

Optimizing the Lagrange function over x, y (primal variables) and μ (dual variables) is a simpler problem and admits an intuitive interpretation since the caching and the bandwidth decisions of the MNO are decoupled. Particularly, the problem can be solved in an iterative fashion [82]. In each iteration, denoted with t, the variables of the dual problem (μ) are updated and accordingly the primal problem is solved in order to update the primal variables (x, y, p^C, p^B), which in turn are used in the subsequent dual objective update. In the next chapter, the primal-dual method is presented for a similar joint caching and routing problem. Hence, its complete description is omitted from this chapter.

3.6.4 Dataset-driven Evaluation

In this subsection, the impact of employing WiFi APs for offloading mobile data is evaluated using the dataset of real residential users. It is shown that MNOs can reduce the cost for serving the mobile users by implementing the proposed mechanism, but the benefits heavily depend on the population density and the cellular network cost in the regional market. In the best scenario, a MNO can improve its bottom line by a factor of 2, while reimbursing up to 9 euros per month each residential user.

The evaluation is carried out for a typical circular-shaped macrocell with radius 2km. The $N = 167$ residential users in the dataset are uniformly placed in random statistically

independent positions in the macrocell. The MNO may employ some of these residential users to offload the macrocell base station (MBS) within the peak-time periods of a month. These are set between 6pm and 11pm each day. Within these periods, each AP can serve the requests for mobile data that are generated within 100 meters distance with an expected bandwidth rate of B_n Mbps which depends on the 802.11 technology used, contention and interference, and is varied in our evaluation.

Next, we describe how to extract the utility coefficient parameters, α_n^B, β_n^B, α_n^S and β_n^S which are introduced in equations (3.25) and (3.26). To start, we make the assumption that each residential user n evaluates the amount of wireless bandwidth that he uses equal or almost equal to the tariff price t_n he currently pays to the provider (Table 3.2). Particularly, we assume that the evaluation of user n is $t_n + v$, where v is a random value in $[-3, +3]$ euros. This is reasonable, since the user would probably have picked a different tariff if this were not the case. Then, replacing the left side of equation (3.25) with $t_n + v$, we obtain that:

$$t_n + v = \alpha_n^B \log(\beta_n^B \cdot r_n) \tag{3.36}$$

where $r_n \in (0, 1)$ captures the average bandwidth demand of user n as observed within the peak time periods in our dataset.

To obtain a second equation with the two unknowns (α_n^B and β_n^B), we assume that the net utility obtained when user n increases his bandwidth consumption from r_n to 1 portion is very low. This is reasonable, since the user would probably have consumed all the available bandwidth if this were not the case. Hence, we obtain:

$$(t_n + v) \cdot (1 + \delta) = \alpha_n^B \log(\beta_n^B \cdot 1) \tag{3.37}$$

where δ is a small positive constant reflecting the net utility for increasing the used bandwidth portion from r_n to 1. Throughout the evaluation, we set $\delta = 0.1$, although this does not impact the qualitative results of our analysis.

To determine α_n^S and β_n^S, we consider each user owning a personal hard disk of capacity $S_n = 100$ GB (typical for modern laptops) and assume that he evaluates it similarly to the current price of online storage devices, i.e., 2 euros per month [74]. Then, assuming that the slope of the function is similar across users ($\beta_n^S = 100$, $\forall n$), we obtain the α_n^S value by equation (3.26). We further upper bound the portion of leased cache capacity by 0.9, assuming that 10% of it is occupied by permanent files that cannot be erased, e.g., system files.

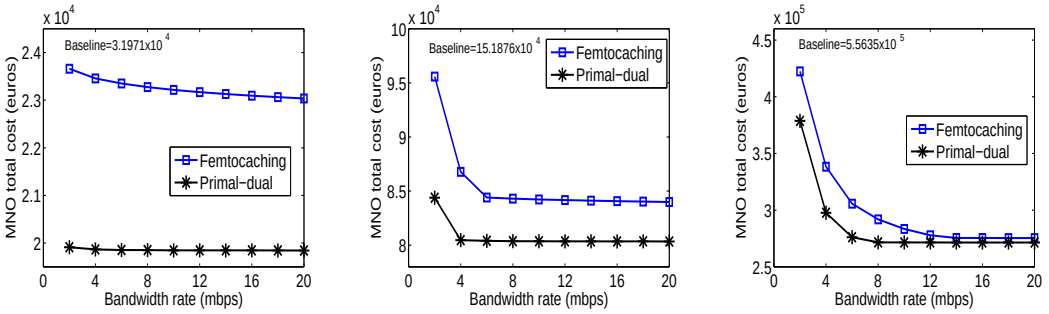
In general, there are several cost components involved in the operation of a cellular network, such as energy consumption, network capacity and infrastructure costs, building and personnel costs, and so on. Apparently, cost structures are typically confidential and not (easily) available to the research community. Besides, many of these costs are managed on slow timescales (over years). The cost function in our model is generic, and hence can apply to all the preceding cases. In the evaluation, we focus on the network capacity cost. Particularly, since the network is usually dimensioned based on the peak load, reducing peak load results in cost savings by postponing the network capacity upgrades needed to meet the growing mobile data demand. Hence, we can approximate the servicing cost by a linear function of the MBS's peak load (as in [75], [76]):

$$Q(y) = s \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} \lambda_{ki} (1 - \sum_{n \in \mathcal{N}_k} y_{ni}^k) \tag{3.38}$$

where $s > 0$ is a scaling constant that is used to convert traffic load into monetary units (euros).

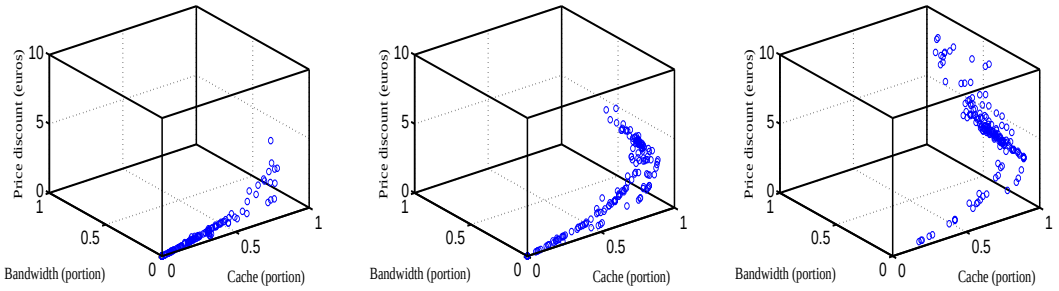
We consider three scenarios differing in the density of mobile user population; 2,000 (small city), 5,000 (sparse city) and 12,000 (dense city) mobile users per km^2 [76]. The slope s is set to 0.0033 /MB in the small city, 0.0063 /MB in the sparse city, and 0.0096 /MB in the dense city; these values are based on AT&T's and Verizon's data plan overage charges in the U.S [76] assuming that one third of the charge covers servicing costs [77]. The mobile users are interested in downloading a collection of $I = 1,000$ popular files, each of size 100MB. On average, a mobile user requests 0.33 files during the peak-time of a day, which results to 1GB of data per month. To capture the spatial heterogeneity of demand, we group mobile users into $K = 200$ classes and randomly place them in the macrocell. Each user class may be covered by more than one distinct APs, but around half of them are covered only by the MBS. Then, we spread the requests across files based on the Zipf model with shape parameter $z > 0$ [61]. Unless otherwise specified, we set $B_n = 9$ mbps [101], $z = 0.8$ [61] and $\pi_n^S = \pi_n^B = 0$, $\forall n \in \mathcal{N}$.

We first look at the impact of varying the bandwidth rate B_n of the APs on MNO's total cost. The latter includes both the cost for providing incentives to residential users and the cost for serving the requests of the mobile users through the MBS. This investigation is important since B_n varies with the 802.11 protocol used, vendors, contention time with other devices on the same frequency, interference, etc [101]. We carry out the investigation separately for the small, sparse and dense city (Figure 3.8). To improve figure visualization, the performance of Baseline (which is independent of the B_n value) is written as text inside the figure instead of being plotted.



(a) Small city (b) Sparse city (c) Dense city

Fig. 3.8: MNO total cost for different AP bandwidth rates in a (a) small, (b) sparse and (c) dense city.



(d) Small city (e) Sparse city (f) Dense city

Fig. 3.9: Scatter plot of monthly price discount (incentive), bandwidth and cache portion leased from residential users in a (a) small, (b) sparse and (c) dense city.

As expected, increasing the AP rate decreases the MNO's total cost for all the algorithms (except from Baseline), since the APs can serve more mobile users. *The offloading benefits are higher in the dense city than in small and sparse*, since the cellular network costs and the demand of mobile users are higher. Consequently, the need for offloading mobile data to WiFi APs is greater. This observation is consistent with the fact that offloading mechanisms have existed for a long time in densely populated areas. The proposed Primal-dual algorithm consistently outperforms a common state of the art algorithm, named Femtocaching [33]. The gains are up to 19%.

We now examine mobile data offloading from the viewpoint of individual residential users. The three-dimensional scatter plots in Figure 3.9 illustrate the distribution of the discounts in monthly subscription bills (incentives), and the portion of leased cache and bandwidth capacity across all the residential users, when Primal-dual algorithm is applied. The results uncover a wide disparity in the amount of leased resources across residential users, especially in the dense city scenario. Also, some of the residential users see a notable reduction in their monthly bills. These generally correspond to the

users residing in areas with relatively high mobile data demand, where the need for offloading mobile data and managing cellular network costs is crucial. In overall, *the price discounts are higher in the dense city scenario, and can reach* 9 *euros per month in the upper extreme case.*

Main takeaways: The facility location and the caching problems are connected. Hence, the algorithms known for the former problem can be used to solve the latter problem. The presented numerical results show that a facility-location inspired caching algorithm in a small-cell network performs up to 38% better that existing caching algorithms. Leveraging the idle residential user-owned resources (cache space and WiFi bandwidth) to store and deliver content to mobile users can further reduce the costs of network operators.

Chapter 4

Caching Layered Video

Contents

4.1 Introduction

Nowadays there is a tremendous growth in the number of mobile users viewing videos [1], which are encoded and pre-stored on servers and delivered over cellular networks. Mobile network operators (MNOs) strive to serve these massive requests and achieve the minimum possible video delivery delay. This is very important since it is the main criteria for the users' perceived satisfaction. However, delivering this content puts unprecedented

pressure on the networks and often yields a very high servicing cost for the operators. Achieving the right balance between this cost and the delivery delay experienced by the users is currently one of the most important challenges for the MNOs.

In this chapter, we revisit the caching problem in Heterogeneous Cellular Networks (HCNs), with cache-endowed small-cell base stations (SBSs), by considering the particular characteristics of video delivery. Specifically, each video file should be available in various qualities since users often have different (minimum) quality requirements (e.g., spatial resolution, frame-rate, etc.). To achieve this, every video can be encoded into multiple versions which differ in quality and rate (*versions*). Another option is scalable video coding (SVC) (*layers*) where each video is encoded into different layers which, when combined, produce a quality that increases as more layers are used. This technique introduces an encoding overhead but offers network flexibility since the layers of each file can be cached at different base stations and/or routed over different paths. The MNO can use versions, layers or a mixture of them for the video files.

Obviously, the MNOs have a large repertoire of video encoding, caching and routing decisions for servicing the user requests. The MNO should take these decisions in a way that balances a cost-performance objective. We propose such an optimization framework, and investigate numerically the impact of caching decisions on the balanced delay and servicing cost. We find that when the user demand is homogeneous in terms of requested video quality, the operator can improve his balanced objective by using versions instead of layered encoding. However, as the users' demand becomes more diverse, layered encoding can be more beneficial, as it allows for more flexible caching and routing decisions. Moreover, we characterize the delay - cost tradeoff. We find that improving the delivery delay (by tuning properly a balancing parameter) by an average 10% may increase the servicing cost from 10% up to 30% depending on the load of the network (users' requests).

We also design caching algorithms for multiple operators that cooperate by pooling together their co-located caches, in an effort to aid each other, so as to avoid large delays that follow from downloading layered video from distant servers. We derive novel approximation algorithms using a connection to a knapsack-type problem and a cache-partition technique. The results demonstrate up to 25% delay savings over existing schemes.

Summarizing, the contributions of this chapter are as follows:

- *Optimization Framework.* We introduce a framework for the joint optimization of video encoding, caching and routing decisions, that minimize a balanced objective of average delay and servicing cost. Our model considers realistic aspects of HCNs

such as the capacitated backhaul links and the constraints for the cache sizes and the wireless capacity of the SBSs.

- *Video Encoding Policies.* We explicitly model and study the impact of the employed video encoding scheme (versions or layers) on the servicing cost and delay. We explain under which conditions the operator should select one of them or even employ both of them. Also, we discuss how our analysis accounts for the video streaming model requirements.

- *Performance Evaluation.* Our study is generic which allows us to investigate the impact of several system parameters. Based on our numerical analysis, we conclude that video encoding decisions are affected by the homogeneity of users requests, the capacity of the SBSs and the encoding overhead of layering. Also, we show that improving the delay up to 10% may induce additional servicing cost which can reach 30% when the network is heavily loaded.

- *Cooperation among operators.* We design caching algorithms for many operators that cooperate in order to cache layered video destined also for users of other operators. Numerical results based on a measured trace of layer sizes demonstrate up to 25% reduction in delay over existing (layer-agnostic) caching schemes.

The rest of this chapter is organized as follows. Sec. 4.2 introduces the system model and the problem. In Sec. 4.3 we solve the video delivery problem for the case of versions encoding. We extend our methodology for the case that both versions and layers are used and discuss the implications for video streaming in Sec. 4.4. Sec. 4.5 provides performance evaluation results. Finally, Sec. 4.6 provides caching algorithms for cooperating operators.

4.2 System Model and Problem Statement

4.2.1 System Model

The system architecture is depicted in Fig. 4.1. We study the downlink operation of an HCN macrocell with one macrocellular BS (MBS), hereafter indexed M, a set (tier) $\mathcal{N}_\mathcal{P} \triangleq \{1, 2, \ldots, N_P\}$ of $|\mathcal{N}_\mathcal{P}|$ picocell base stations (PBSs) covering smaller areas within the macrocell, and a set $\mathcal{N}_\mathcal{F} \triangleq \{1, 2, \ldots, N_F\}$ of $|\mathcal{N}_\mathcal{F}|$ femtocell base stations (FBSs) with transmission range of few tens of meters[1]. We denote as $\mathcal{N} \triangleq \mathcal{N}_\mathcal{P} \cup \mathcal{N}_\mathcal{F}$ the set of all small cell base stations (SBSs).

[1]The analysis can be extended for more classes of small cell base stations (e.g., microcells), and can be generalized for multiple macrocells.

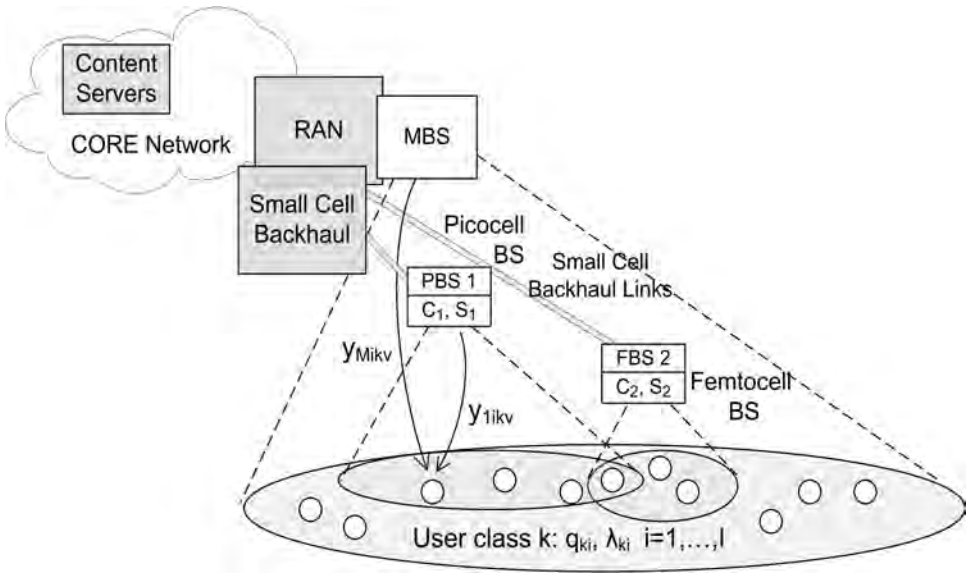


FIGURE 4.1: A multi-tier cellular architecture with store-capable base stations.

We study the system for a certain time period during which each SBS $n \in \mathcal{N}$ has an average wireless capacity of $C_n \geq 0$ bps, while the capacity of the macrocell is $C_M \geq 0$ bps. BSs of the same type have similar characteristics but may differ in certain cases (e.g., due to location-dependent shadowing effects). The MBS coverage overlaps with all the other base stations, while it is also possible to have overlapping femto or picocell base stations. We consider *disjoint subchannel allocation* among different tiers of BSs as in the previous chapter. Neighboring BSs in the same tier can also be assigned orthogonal frequency bands or employ enhanced inter-cell interference coordination techniques (eICIC) proposed in LTE Rel. 10.

Each SBS $n \in \mathcal{N}$ is connected to the core network through a wired or wireless backhaul link of average capacity $G_n \geq 0$ bps. These links connect the SBS to certain aggregation points, e.g., fibre cabinets close to macrosites. The MBS is connected to the core network through the typical high-capacity RAN backhaul. Finally, each SBS n is endowed with a certain storage capacity of $S_n \geq 0$ bytes.

We study the delivery of a large set $\mathcal{I} \triangleq \{1, 2, \ldots, I\}$ of video files, each one of which can be delivered in $Q > 1$ different quality levels. The term quality level in this chapter can correspond to different spatial resolutions (frame sizes), different temporal resolutions (frame rates), or different SNR qualities (controlled at the video coder). We assume that there is a set $\mathcal{V}$ of versions that can be offered for each file $i \in \mathcal{I}$. Each version $v \in \mathcal{V}$ corresponds to a certain quality level (it is $|\mathcal{V}| = Q$) and has size o_{iv} bytes which increases with the quality, i.e., $o_{iv} \geq o_{iu}$ if $v > u$ (assuming there is an ordering in versions wrt their quality and size).

Also, we assume that there is a set of layers $\mathcal{L}$ that can be offered for each video file when it is encoded with scalable video coding (SVC). This is an extension of the H.264/MPEG4-AVC standard that offers, among others, quality scalability [78]. A video decoder can reconstruct the video sequence by receiving a subset of them. In order to decode the layer l, all preceding layers $l' \leq l$ of the same video file should be available[2]. Let o_{il} denote the size of layer $l \in \mathcal{L}$ of file i. Compared to versions, layered encoding typically incurs an encoding overhead:

$$\sum_{l=1}^{q} o_{il} = o_{iv}(1 + R_i^q), \ v = q, \ \forall q \in \{1, 2, ..., Q\} \tag{4.1}$$

where $R_i^q > 0$ is the encoding overhead for the quality level q of file i that can be calculated with experimental methods [79]. Layered encoding through SVC has been used extensively the last few years in the real-world video-conferencing systems[3].

We model demand by introducing a set $\mathcal{K} \triangleq \{1, 2, \ldots, K\}$ of user classes, each one representing a subset of users in the same location (very small subregion of the macrocell), asking for (possibly) different files with certain minimum quality requirements[4]. Video files are delivered through the streaming mechanism. This means that the user starts decoding and rendering the video file before it receives it in its entirety. This aspect is analyzed in detail in Section 4.4.B. User locations can be random, e.g., following a uniform distribution. Let $\lambda_{ki} \geq 0$ denote the demand (i.e., number of requests) of user class k for file $i \in \mathcal{I}$ with minimum quality $q_{ki} > 0$. This means that user class k should get either one version v with $v \geq q_{ki}$, or all layers up to q_{ki}, i.e. $l = 1, 2, \ldots, q_{ki}$. Unless otherwise specified, a user requesting (minimum) quality q_{ki} can be served with higher quality as well. The total demand that must be served by the MNO for the specific macrocell is:

$$\Lambda = \big((\lambda_{ki}, \, q_{ki}) \, : \, k \in \mathcal{K}, \, i \in \mathcal{I}\big) \tag{4.2}$$

We denote with $\mathcal{N}_k \subseteq \mathcal{N}$ the subset of BSs that are in range with user class k and assume that user association can be accomplished based on network performance or cost criteria.

4.2.2 Problem Statement

The objective of the network operator is to deliver as many of the requested video files as possible, with the minimum delay and the minimum possible servicing cost. These latter

[2]The ordering is wrt quality: with slight abuse of notation, we use the index of the layer (and the version) to denote also the respective quality.

[3]See for example, Vidyo: http://www.vidyo.com, and Radvision: http://www.radvision.com.

[4]Notice that this is not a restriction or an assumption, rather it implies that we group the users based on location and on quality requirements.

depend on the demand and the servicing policy of the operator, i.e., the base stations and the backhaul links that will be used. We denote with $J_n(a) \geq 0$ the cost incurred by the MNO when it uses SBS $n \in \mathcal{N}$ to deliver content with rate of $a \geq 0$ bps, and $J_M(a)$ the respective cost for the MBS. This cost includes the BS energy consumption and is positively correlated to the distance between the BS and the served users.

Since the resources of the operator are limited, some user requests may not be served or served with practically intolerable delays. This induces cost to the operator due to future revenue losses, e.g., unsatisfied clients unsubscribe from the service. We introduce a penalty function $P(\cdot)$ to capture this cost, which is assumed to be a positive, increasing and convex function of the number of unserved requests.

The average delay D_{ki} experienced by user class k for downloading item $i \in \mathcal{I}$ depends on the path and the congestion of the respective links. The main components of the delay are the processing, propagation and transmission delay as well as the queueing delay which captures link congestion [80]. For each user class k the operator determines the portion of the requests that will be routed over each possible path leading either to a cache of an SBS (having the item), or to a content server. Clearly, the routing decisions of the operator are coupled with its caching policy.

The MNO decides whether it will cache a certain file and with what quality at each SBS. Different versions (or layers) of each file have different size and can satisfy different subset of requests. In order to serve a user by a SBS, the requested content should either be already cached there or fetched via the respective backhaul link. This latter option adds delay which, depending on the backhaul type, capacity and load, may be quite significant. Alternatively, the MBS can serve the requests with smaller delay (if it is not heavily loaded).

The delay minimization and the cost minimization may in general be conflicting objectives, and hence need to be balanced properly, depending on the objective of each operator. Formally, the problem of the MNO can be defined as follows.

MNO Video Delivery Problem (MVD). *Given: (1) the matrix Λ of requests for video files, for a certain time period, (2) the storage and average capacities of the SBSs, S_n, C_n, G_n, $\forall n \in \mathcal{N}$, (3) the servicing cost of the BSs, $J_M(\cdot)$, $J_n(\cdot)$, $\forall n \in \mathcal{N}$, and (4) the penalty cost $P(\cdot)$ for rejecting requests:*

- *for each video file, decide in which base stations it will be cached and at which quality (which version/layers),*

- *for each request from user class k, determine from which base stations it will be served, whether backhaul links will be used or if it will be delivered by the MBS, and with what quality,*

so as to optimize the balanced objective of average user experienced delay, and total servicing and penalty cost.

4.3 Delivering Versions

In this section we formally introduce the problem for the case of video encoding in multiple qualities (versions) and accordingly present a methodology for its solution.

4.3.1 MVD Problem Formulation

Decision Variables and Constraints. Let $x_{niv} \in \{0,1\}$ denote whether version $v \in \mathcal{V}$ of file $i \in \mathcal{I}$ will be cached at SBS $n \in \mathcal{N}$. These variables consitute the caching policy of the MNO:

$$x = \big(x_{niv} \,:\, n \in \mathcal{N},\, i \in \mathcal{I},\, v \in \mathcal{V}\big) \tag{4.3}$$

Also, the variable $y_{kniv} \in [0,1]$ denotes the portion of requests of user-class k for file i that will be satisfied with version v downloaded from SBS $n \in \mathcal{N}$, and $y_{kMiv} \in [0,1]$ is the respective decision for the MBS (M). Finally, $z_{kniv} \in [0,1]$ is the portion of requests of user k for file i that require fetching version v via the backhaul of SBS $n \in \mathcal{N}$. The routing policy of the MNO is described by the following matrices:

$$y = \big(y_{kniv},\, y_{kMiv} \,:\, k \in \mathcal{K},\; n \in \mathcal{N}_k,\; i \in \mathcal{I},\; v \in \mathcal{V}\big) \tag{4.4}$$

$$z = \big(z_{kniv} \,:\, k \in \mathcal{K},\, n \in \mathcal{N}_k,\, i \in \mathcal{I},\, v \in \mathcal{V}\big) \tag{4.5}$$

In order for a SBS to send a version to a user, it needs either to have it cached or to download it through the backhaul. Moreover, the servicing rate from SBS n to user k can be maximum if item i is cached, and if not, it cannot exceed backhaul servicing rate. Hence:

$$y_{kniv} \le x_{niv} + z_{kniv} \;\; \forall\, k \in \mathcal{K},\, n \in \mathcal{N}_k,\, i \in \mathcal{I},\, v \in \mathcal{V} \tag{4.6}$$

Besides, the MNO cannot satisfy a request more than once:

$$\sum_{v \ge q_{ki}} y_{kMiv} + \sum_{n \in \mathcal{N}_k} \sum_{v \ge q_{ki}} y_{kniv} \le 1, \;\; \forall\, k \in \mathcal{K},\, i \in \mathcal{I} \tag{4.7}$$

Notice that the requests of each user class for each file may be satisfied by versions of different qualities (higher or equal to the minimum quality required by the user). Hence, there is a need to add the respective components.

Clearly, the routing and caching policies are constrained by the wireless capacity and storage capacity of the base stations. Also, for the routing decisions there may be additional constraints due to interference. The current industry practice is to use a disjoint channel allocation across different tiers of base stations, as well as orthogonal subchannels for overlapping base stations within each tier. Channel orthogonality across different BSs can also be provided with the introduction of almost blank subframes (ABS) that were defined in the eICIC mechanism. The impact of this type of interference management techniques on the system's (average) capacity can be modeled using the protocol interference model. Clearly, the data delivery of interfering SBSs cannot be concurrently maximized. Specifically, it should hold:

$$\sum_{n\in\mathcal{N}_k}\frac{1}{C_n}\sum_{m\in\mathcal{K}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} y_{mniv}o_{iv}\lambda_{mi}\leq 1,\ \forall\, k\in\mathcal{K} \tag{4.8}$$

where $\mathcal{N}_k$ is the set of SBS in range with user k and hence are potential interferers.

Delay-Cost Minimization. The delay cost of user class k requesting file i, depends on caching x and routing policy (y,z):

$$D_{ki}(y,z,\Lambda)=\sum_{v\geq q_{ki}}\big(d_{kMiv}(y,\Lambda)+\sum_{n\in\mathcal{N}_k} d_{kniv}(y,z,\Lambda)\big) \tag{4.9}$$

where d_{kniv} is the delay when users k are served by SBS $n\in\mathcal{N}_k$ with version v of file i, and d_{kMiv} is the delay when served by the MBS. Notice that the summation over the different versions is necessary since a request can be satisfied with higher quality version, e.g., if it is already available in a nearby SBS. The flow-level delay we consider here has fixed components, such as the propagation and the processing delay at the base stations (or routers), and some components that vary with the load of the links. The latter is often modeled as a queueing delay under the hypothesis of $M/M/1$ queueing behavior [80]. Namely, each bit traversing a link of capacity C bps which carries a total flow with rate $f\geq 0$ bps, experiences a delay of $1/(C-f)$ seconds.

The MNO servicing cost $J_n(y,z,\Lambda)$ depends on the total bandwidth that each SBS n delivers to the subscribers, including the backhaul link consumption. Similarly, for the MBS, it is $J_M(y,\Lambda)$. Notice that we do not take into account the RAN backhaul cost since this hop is common for the SBS and the MBS. We assume that these functions are positive, increasing and convex on the delivered bandwidth due to congestion effects

[81][5]. The aggregate servicing cost of the operator can be defined as follows:

$$J(y,z,\Lambda) = J_M(y,\Lambda) + \sum_{n\in\mathcal{N}} J_n(y,z,\Lambda) \tag{4.10}$$

The goal of the MNO is to minimize, for a certain time period of duration T, the average delay for all users and the total servicing and penalty cost. This is achieved by solving the following joint routing and caching optimization problem (*MVD Problem*):

$$\min_{x,y,z} J(y,z,\Lambda) + P(y,\Lambda_P) + \alpha \sum_{k\in\mathcal{K}} \sum_{\in\mathcal{I}} D_{ki}(y,z,\Lambda)$$

s.t. (4.6), (4.7), (4.8)

$$\sum_{i\in\mathcal{I}} \sum_{v\in\mathcal{V}} o_{iv} x_{niv} \leq S_n, \ \ \forall\, n \in \mathcal{N} \tag{4.11}$$

$$\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \sum_{v\geq q_{ki}} \lambda_{ki} o_{iv} y_{kniv} \leq C_n T, \ \ n \in \mathcal{N} \tag{4.12}$$

$$\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \sum_{v\geq q_{ki}} \lambda_{ki} o_{iv} z_{kniv} \leq G_n T \ \forall\, n \in \mathcal{N} \tag{4.13}$$

$$\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \sum_{v\geq q_{ki}} \lambda_{ki} o_{iv} y_{kMiv} \leq C_M T \tag{4.14}$$

$$z_{kniv}, y_{kniv}, \ y_{kMiv} \in [0,1], \ \forall k \in \mathcal{K}, n \in \mathcal{N}_k, i \in \mathcal{I}, v \in \mathcal{V} \tag{4.15}$$

$$x_{niv} \in \{0,1\} \ \ \forall\, n \in \mathcal{N}, \ i \in \mathcal{I}, \ v \in \mathcal{V} \tag{4.16}$$

where parameter $\alpha > 0$ (measured in monetary units over time units) is determined by the operator and is used to balance the cost-delay objectives. For example, an operator interested in reducing the delay even at the expense of higher servicing cost may set a high value for α. Also, there is no MBS backhaul constraint (assume that RAN backhaul is sufficiently large) and it is $y_{kniv} = 0$ when $n \notin \mathcal{N}_k$. Finally, Λ_P is the number of unserved requests:

$$\Lambda_P = \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \lambda_{ki} \big(1 - \sum_{v\geq q_{ki}} [y_{kMiv} + \sum_{n\in\mathcal{N}_k} y_{kniv}]\big)$$

The objective function of the MVD problem is convex under the assumptions about the properties of $D_{ki}(\cdot)$, $J(\cdot)$ and $P(\cdot)$. However, the constraint set includes the $0-1$

[5]The exact form of the backhaul link cost function depends on various technical parameters (e.g., transmission technology) and economic parameters (leased or owned) [81].

decisions variables x_{niv} that render it *NP-hard*. In the sequel, we present an algorithm for deriving a solution that asymptotically converges to the optimal one.

4.3.2 MVD Solution Method

In order to solve the MVD problem we use the method of Lagrange partial relaxation [82]. Specifically, we relax the constraints in (4.6) and introduce the respective set of dual Lagrange multipliers:

$$\mu = (\mu_{kniv} \geq 0 : \forall\, k \in \mathcal{K},\, n \in \mathcal{N}_k, i \in \mathcal{I}, v \in \mathcal{V}) \tag{4.17}$$

This relaxation simplifies the solution of the problem and admits an intuitive interpretation since it decouples the routing and the caching decisions of the operator.

First, we define the Lagrange function as follows:

$$\begin{aligned} L = J(y,z,\Lambda) + P(y,\Lambda_P) + \alpha \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} D_{ki}(y,z,\Lambda) + \\ + \sum_{k\in\mathcal{K}} \sum_{n\in\mathcal{N}_k} \sum_{i\in\mathcal{I}} \sum_{v\in\mathcal{V}} \mu_{kniv}(y_{kniv} - x_{niv} - z_{kniv}) \end{aligned}$$

Using this relaxation, the problem can be rewritten:

$$\begin{aligned} \max_{\mu} \min_{x,y,z} \quad & L(x,y,z,\mu) \\ s.t. \quad & (4.7), (4.8), (4.11), (4.12), (4.13), (4.14), (4.15), (4.16) \\ & \mu_{kniv} \geq 0,\, \forall k \in \mathcal{K}, n \in \mathcal{N}_k,\, i \in \mathcal{I}, v \in \mathcal{V} \end{aligned} \tag{4.18}$$

which can be solved in an iterative fashion, using a primal-dual Lagrange method. Notice that due to the discrete constraint set, we have to employ a subgradient method for updating the dual variables [82].

In each iteration, denoted with t, the dual objective is improved using a subgradient update and accordingly the primal relaxed problem is solved in order to update the primal variables (which in turn are used in the subsequent dual objective update). The dual variables are updated with a subgradient method as follows:

$$\mu_{kniv}^{(t+1)} = [\mu_{kniv}^{(t)} + \sigma^{(t)} g_{kniv}^{(t)}]^{+},\, \forall\, k \in \mathcal{K}, n \in \mathcal{N}, i \in \mathcal{I}, v \in \mathcal{V} \tag{4.19}$$

where $[.]^{+}$ denotes the projection onto the non-negative orthant, and $\sigma^{(t)}$ is the step size at iteration t. Also, $g_{kniv}^{(t)}$ is the subgradient, i.e., $g_{kniv}^{(t)} = y_{kniv}^{(t)} - x_{niv}^{(t)} - z_{kniv}^{(t)}$.

Algorithm 4.1: Primal-Dual Algorithm

Input : $J(\cdot)$, $P(\cdot)$, $D_{ik}(\cdot)$, α, Λ
Output: x^*, y^*, z^*

Initialize dual variables μ^1 to zero, the lower bound as $LB = -\infty$ and the upper bound as $UB = +\infty$.
$t \leftarrow 1$;
repeat
  Solve P_1 and find $x^{(t)}$;
  Solve P_2 and find $y^{(t)}$ and $z^{(t)}$;
  Name as $q(\mu^{(t)})$ the solution value of the primal problem;
  if $q(\mu^{(t)}) > LB$ **then**
    $LB = q(\mu^{(t)})$;
  Update UB;
  Update dual variables $\mu^{(t+1)}$ using (4.19);
  $t \leftarrow t + 1$;
until $\frac{UB-LB}{UB} < 0.1$ *and* $t < 1000$;

Then, we need to solve the relaxed primal problem and obtain the updated values of x, y and z (for the current iteration t). Interestingly, the primal problem can be further decomposed into two subproblems, named P_1 and P_2, as follows:

$$\begin{aligned} P_1 \quad &: \max_{x} \sum_{k\in\mathcal{K}} \sum_{n\in\mathcal{N}_k} \sum_{i\in\mathcal{I}} \sum_{v\in\mathcal{V}} \mu_{kniv} x_{niv} \\ s.t. \quad & \quad (4.11), (4.16) \end{aligned}$$

and

$$\begin{aligned} P_2 \quad &: \min_{y,z} J(y,z,\Lambda) + P(y,\Lambda_P) + \\ &+\alpha \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} D_{ki}(y,z,\Lambda) + \\ &+\sum_{i\in\mathcal{I}} \sum_{k\in\mathcal{K}} \sum_{n\in\mathcal{N}_k} \sum_{v\in\mathcal{V}} \mu_{kniv}(y_{kniv} - z_{kniv}) \\ s.t. \quad & \quad (4.7), (4.8), (4.12), (4.13), (4.14), (4.15) \end{aligned}$$

P_1 involves only the caching variables x. Hence, we call it the *caching subproblem*. Also, P_1 is separable into $|\mathcal{N}|$ unidimensional knapsack problems, one for each SBS $n \in \mathcal{N}$, and thus can be solved in a distributed manner. The knapsack problem can be optimally solved in pseudo-polynomial time using dynamic programming methods. On the other hand, P_2 involves only the bandwidth allocation decisions y and z. We call it the *bandwidth allocation* subproblem. The objective function of P_2 is strictly convex and the constraint set is convex, compact and continuous. Hence, it can be efficiently

solved using standard convex optimization techniques [82]. The method is summarized in Algorithm 4.1.

The solution that we provide for this NP-hard problem converges asymptotically to the optimal solution. Specifically the following lemma holds:

Lemma 4.1. *Algorithm 4.1 converges asymptotically to the optimal solution* x^*, y^*, z^*.

Proof. The convergence of this type of primal-dual algorithms with subgradient updates (non-differentiable dual functions) is ensured if (i) a proper diminishing step size is selected satisfying conditions of Prop. 8.2.6 [82, Chapter 8], (ii) the subgradients are bounded. Here, we follow the methodology in [83] and set $\sigma^{(t)} = \nu \frac{UB - q(\mu^{(t)})}{||g^{(t)}||^2}$, where ν is a parameter with positive value and UB is the upper bound on each iteration that can be calculated by simply finding a feasible solution to the primal problem. Also, by their definition, it can be directly seen that the subgradients are bounded. □

An operator can use Algorithm 4.1 to solve offline, that is at the beginning of each time period, the MVD problem and find a near optimal joint routing and caching policy for versions. In the sequel, we extend our methodology for the case of layered encoding.

4.4 Delivering Layers and Video Streaming

In this section, we explain how the system architecture and the video delivery optimization problem described previously change, when the MNO employs SVC for compressing and delivering the video files. Next, we discuss how the proposed optimization approach is related to the key video streaming parameters that are configured at the video decoder.

4.4.1 Layered Encoding

We extend the model to include video encoded in layers, and more specifically we adopt the SVC extension of the H.264/MPEG4-AVC standard [78]. The operator can deliver either a version or a subset of layers that satisfies the minimum quality requirement of user requests. One of our goals is to investigate under what conditions caching and delivering layers is less costly for the operator than delivering versions of the files. This is of major importance since layers are rarely used today by CDNs (and MNOs) due to the additional rate overhead they introduce. However, as we will explain in the sequel and demonstrate in the performance evaluation section, for HCNs with storage capable SBSs, caching layers may be beneficial in terms of servicing cost and experienced delay

instead of transcoding the video into multiple versions. The reason is that more flexible caching and routing decisions can be made, since each user can receive different layers (for the same file) through different paths. For example, the base layer can be delivered even via costly links if this ensures small delays, while subsequent layers (that improve quality) can be delivered through other, less costly and/or congested paths.

Before we formally describe the extension of our optimization framework, we have to clarify that SVC allows different types of scalable encoding (spatial, temporal, quality) to be combined and create a single layer. Our modeling approach for layered video is generic and it can capture the delivery of all the potential scalability combinations that are available in SVC. This is possible because every scalability combination can be expressed in terms of specific well-ordered dependencies between the involved layers. In this chapter we are not concerned with the specifics of the layered encoding and the optimal selection of scalability combinations but only with the implications on network delivery of this relatively new type of encoded video streams.

The set of layers is denoted as $\mathcal{L} \triangleq \{1, 2, \ldots, Q\}$. These layers introduce additional constraints on the derivation of the MNO's policies. Namely, there is no benefit for a user to receive a specific layer if it has not received all the lower layers. Hence, for each user k requesting file i should hold:

$$\sum_{n \in \mathcal{N}_k} y_{kni(l+1)} = \sum_{n \in \mathcal{N}_k} y_{knil}, \ \forall\, l < q_{ki} \tag{4.20}$$

$$\sum_{n \in \mathcal{N}_k} y_{knil} = 0, \forall\, l > q_{ki} \tag{4.21}$$

where we extended the definition of y variables, so that they also refer to layers, i.e. $y_{knil} \in [0, 1]$ denotes the portion of requests of user-class k for file i that will be satisfied with layer l downloaded from BS $n \in \mathcal{N}$. We do the same for the rest of the optimization variables x and z.

Requests can be satisfied either using layers or versions. In any case, each request by user k for each file i cannot be satisfied more than one time:

$$y_{kMil_q} + \sum_{v \geq q_{ki}} y_{kMiv} + \sum_{n \in \mathcal{N}_k} y_{knil_q} + \sum_{n \in \mathcal{N}_k} \sum_{v \geq q_{ki}} y_{kniv} \leq 1 \tag{4.22}$$

where $l_q \in \mathcal{L}$ is the highest layer required to satisfy quality request q_{ki}.

The delay components are similarly defined as before, with the only difference that there is a need to minimize the *maximum delay* for each delivered layer (since all layers must be delivered in order to watch the video). Namely, the delay D_{ki} is now written as

follows:

$$\begin{aligned} D_{ki}(y,z,\Lambda) &= \sum_{v \geq q_{ki}} (d_{Mikv}(y,\Lambda) + \sum_{n \in \mathcal{N}_k} d_{nikv}(y,z,\Lambda)) \\ &+ d_{kMil_d}(y,\Lambda) + \sum_{n \in \mathcal{N}_k} d_{knil_d}(y,z,\Lambda) \end{aligned} \tag{4.23}$$

where l_d is the layer that is delivered with the largest delay (for each user k and each file i). Notice also that some requests of each user for each file may be satisfied by layers while others by versions. Hence, there is a need to add the respective delay components.

4.4.2 Video Streaming Concerns

From the perspective of the end user the result of our optimization is the minimized delivery delay of the requested video file. This delay will directly impact the video streaming process through the necessary *startup delay* that is introduced at the video decoder of the user. In the sequel, we will describe how the benefits that our optimization framework provides can be readily translated into a minimized startup delay.

In typical video decoders there are two delay components that are introduced before the video playback commences. First, for playback to start there is a need to buffer a certain number of video frames that can be translated to either a portion of the file in bytes or seconds. Let us denote this inelastic buffering delay as D_{b_1}. Second, the video decoder usually adds an additional delay element before it starts the playback in order to accommodate fluctuations in the bandwidth of the network. This is usually exercised by measuring the RTT and the average receiver data rate. This additional delay component is denoted as D_{b_2}. Finally, when the video decoder commences the playback of the video, it will normally maintain a constant playback rate in terms of frames-per-second (fps). Thus, this delay component that accounts for the rendering of the video file is denoted as D_r and is simply equal to the length of the video in seconds. What all the above mean is that once the user requests the video file, say at time instant $t = 0$, the time instant that the playback ends is equal to $D_{b_1} + D_{b_2} + D_r$.

From the last three delay components we discussed the only one that can be practically controlled is D_{b_2}. If at the decoder this delay is too small then the decoder might experience a *buffer underrun* which means that it requires data for decoding and playback but they have not yet been received. This is typically addressed with the undesired playback pauses. To avoid these events, the average delay that the complete video is delivered, must be lower than the time instant that the playback ends at the user decoder. Thus,

the following condition must hold:

$$D_{ki}(y, z, \Lambda) \leq D_{b_1} + D_{b_2} + D_p \tag{4.24}$$

Thus, by minimizing the file delivery delay $D_{ki}(y, z, \Lambda)$, we can indirectly allow the video decoder to use a smaller delay D_{b_2}. For the defined system model, our optimization framework ensures the minimum startup delay in order to avoid a buffer underrun (for a specific cost-delay tradeoff expressed with parameter α).

Also, it is clear that the video streaming performance depends on the quality level of the delivered video, as this in turn determines the video file size. Hence, delivering versions of high quality is more likely to induce buffer underrun or other similar undesirable phenomena. On the other hand, layered encoding has a unique advantage as one can determine the video quality more dynamically. For example, deliver first the basic layer (with small size), and then, if the network conditions change, deliver the higher quality layer. This is particularly important for the case that the routing and caching decisions are taken not for the entire video files but for the video segments.

4.5 Performance Evaluation

In this section we present the numerical experiments that we have conducted to evaluate the performance of the proposed algorithm using realistic system settings. Our main objectives/goals are as follows:

- Compare the two different video encoding techniques, i.e. layered encoding and versions.
- Describe the cost-delay trade off.
- Examine the convergence of the proposed algorithm.

Methodology and Performance Criteria. Particularly, we compare the performance of our algorithm in three cases:

1. *Versions*: The files are encoded with different rates that yield multiple versions.
2. *Layered-Encoding*: Each file has a multi-layer representation based on SVC.
3. *Mixed Strategy*: The system supports both versions and layers.

The performance criteria that we consider are the incurred cost by the operator and the experienced delay by the users. Because of the quadratic relation between power consumption and achievable rate, we adopt the following cost function:

$$J_n(y,z,\Lambda) = \big(\sum_{k\in\mathcal{K}} w_{kn} \sum_{i\in\mathcal{I}} \sum_{q\in\mathcal{V}\cup\mathcal{L}} o_{iq}\lambda_{ki} y_{kniq}\big)^2 + \\ + \beta\big(\sum_{k=1}^{K}\sum_{i=1}^{I} \sum_{q\in\mathcal{V}\cup\mathcal{L}} o_{iq}\lambda_{ki} z_{kniq}\big)^2$$

where β is a positive constant and $w_{kn} \geq 0$ captures the wireless transmission efficiency among user k and base station n. The largest the value of w_{kn} is the less are the resources the network has to consume (e.g., frequency - time slots, or energy) in order to serve the user[6]. The experienced delay for each user k receiving version (or layer) q of file i by BS n is [84]:

$$d_{kniq}(y,z,\Lambda) = \frac{y_{kniq}\lambda_{ki}o_{iq}}{C_n - A_n(y)} + d_n^k y_{kniq}\lambda_{ki}o_{kniq} \\ + \frac{z_{kniq}\lambda_{ki}o_{iq}}{G_n - B_n(z)} + d_n^{bh} z_{kniq}\lambda_{ki}o_{kniq}$$

where $A_n(y) = \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}\sum_{q\in\mathcal{V}\cup\mathcal{L}}(y_{kniq}\lambda_{ki}o_{iq})$ is the assigned load to the BS n. Similarly, for the backhaul link it is $B_n(z)$. d_n^k is the propagation and processing delay for the transmission from base station n to user k, and d_n^{bh} the respective delay component for the backhaul link of the base station.

We adopt the following linear penalty cost function for the unserved requests: $P(y,\Lambda_P) = \gamma\Lambda_P$, where Λ_P is the number of unserved requests, as defined in Section III, and γ is the unit cost incurred per unserved request.

Simulation Setup. Throughout, we consider a cellular network consisted of a single main base station located at the center of a circular-shaped cell with radius $200m$. $K = 200$ mobile users, $|N_{\mathcal{P}}| = 4$ $PBSs$ and $|N_{\mathcal{F}}| = 16$ $FBSs$ are uniformly placed in random statistically independent positions in the cell. The transmission radius of a PBS and a FBS is equal to $80m$ and $50m$ respectively. Neighboring BSs operate in orthogonal frequency bands. The coefficients w_{kn} are set as the fraction of the distance between the user k and the base station n over the radius of the cell. We also set $d_n^k = 1$ and $d_n^{bh} = 1$ for each base station n, $\beta = 1$ and $\gamma = 1000$.

In all simulations, we assume a collection of $M = 100$ files, each one of which can be delivered in $Q = 2$ quality levels. The size of a version of a file is equal to 10 and 20 units

[6]More formally, we can define the transmission efficiency as the average volume of traffic (measured in bits) that can be supported by one unit of spectrum resource (measured in Hz). Obviously, the transmission efficiency is closely related to the path loss and shadow fading of a link. In the simplest case, it can be defined as the Shannon capacity.

of data in the low and the high quality level respectively. The layered-encoding overhead is 10%. This is a realistic choice inspired by the video traces in [85]. Within a period T of size normalized to 1, each user requests a file based on a Zipf-Mandelbrot model [61] with a shape parameter value equal to 0.8 and a shift parameter value equal to 10. Unless otherwise specified, $C_n = 100$, $\forall n \in \mathcal{N} \cup M$, $S_n = 50$, $G_n = 100$, $\forall n \in \mathcal{N}$, $\alpha = 1$ and the requested video quality follows the uniform probability distribution. The parameter ν of Algorithm 4.1 is initially set to 2.0 and is halved if there is no improvement in the UB for 50 successive iterations [83].

Comparison between Versions and Layers. We first compare the balanced cost of the operator achieved by the proposed algorithm as a function of the probability that a user request is for the high quality level in Figure 4.2(a). As expected, this cost increases when the aforementioned probability increases, as more users request the high quality level of the files and, thus, a larger amount of data is downloaded. We observe that *pure version caching is desirable when the requests are homogeneous in terms of the requested quality level*. This is because of the overhead that layered encoding incurs. However, when both the quality levels are requested layered encoding may be preferable, as it requires less storage space for serving the same requests compared to the pure versions scheme. Mixed strategy operates the best, as it efficiently combines the inherent features of the two encoding methods (e.g. the user demand is homogeneous in terms of the requested quality in a cell's subregion, but it is heterogeneous in the rest of it).

We then analyze the impact of the transmission bandwidth capacity of each SBS on the balanced cost of the operator. Figure 4.2(b) shows the results when all the requests are for the high quality level. We observe that for low capacity values the layers achieve lower cost than the pure version scheme. This is because *using layers balances the traffic across the base stations*, as the same user can fetch the two layers from two different neighboring base stations. This cost decrement is more crucial when the base station capacity is low. For higher values of the BS capacity, the layered encoding overhead can not be justified by the above gain and, thus, the pure versions caching scheme outperforms the layered scheme.

The superiority of the layered scheme compared with the versions scheme depends on the encoding overhead, as depicted in Figure 4.2(c). When this overhead becomes large enough, the pure version caching scheme becomes more preferable.

Study of the cost-delay trade off. The objective's balancing parameter α reflects the preference of the operator for reducing the user experienced delay or the servicing cost. Figures 4.2(d)-4.2(e) show the results in the versions case when the per user demand is 1 (Low Load) and 10 (High Load). *As α increases the user delay decreases, at the expense of the servicing cost increase and vice versa*. This is because when α is low, users are

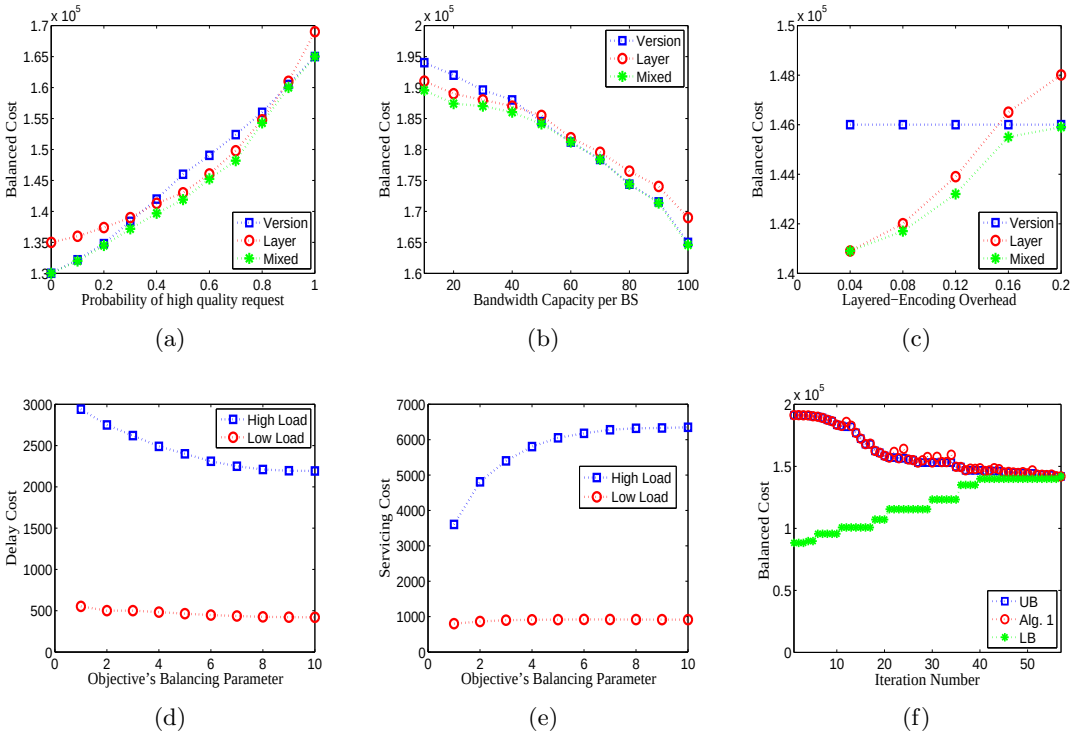


FIGURE 4.2: The balanced cost of the operator as a function of (a) the probability that a request is for the high quality level, (b) the bandwidth capacity per BS and (c) the layered encoding overhead. The impact of the objective's balancing parameter α on (d) the delay cost and (e) the servicing cost. (f) The balanced cost of the operator, the UB and the LB at each step of the execution of Algorithm 4.1.

assigned to the SBSs with the largest spectral efficiency, but when α increases they are forced to be assigned to the less loaded SBSs regardless of the spectral efficiency (and thus consuming more energy). The impact of the parameter α on the two metrics is greater in the High Load case, as the assignment of the users to the overloaded base stations is more costly to the operator. We observe that improving the delivery delay (by tuning properly the parameter α) by an average 10% may increase the servicing cost from 10% up to 30%.

Convergence of Algorithm 4.1. The MVD problem is an NP-hard problem and hence its solution cannot be derived in polynomial time. The iterative solution we propose gradually improves the obtained result. In other words, a network operator can execute the suggested algorithm in an offline fashion to determine for each time period the joint routing and caching policy. The performance improves with the time that the algorithm runs. Specifically, the convergence of our algorithm in the versions case is depicted in Figure 4.2(f). Even when layers come into the picture, *the convergence typically happens in less than a few hundreds steps.*

4.6 Cooperative Caching of Layered Video

Going one step further, we study the performance benefits that may arise when different network operators (NOs) cooperate in SVC caching. Today there exist many market entities (e.g., different Telco-CDNs) that often deploy their own caches in the same locations so as to serve their users-clients. The caches may be interconnected with each other [86]. Thus, it is meaningful to explore the potential of a local cache of a certain network entity to retrieve a video layer from the co-located cache of a different network entity, instead of fetching it from a distant server of its own that would cause larger delay. However, the diverse user demands that the networks must serve render this cooperative caching problem particularly challenging. Hence, it is important to derive a joint caching policy that minimizes the total delay for all network operators, considering the global content demand.

4.6.1 Cooperative Caching Model

We update the notation so that $\mathcal{N}$ represents the set of cache-nodes instead of SBSs, $\mathcal{V}$ the set of videos and $\mathcal{L}$ the set of SVC layers. The size of the l^{th} layer of video v is o_{vl} bytes, which typically decreases with l, i.e., $o_{v1} \geq o_{v2} \geq ... \geq o_{vQ}$. Each cache-node belongs to a NO in the set $\mathcal{K}$ of size K, and is located at a region in the set $\mathcal{M}$ of size M (Figure 4.3). Here, $\mathcal{N}_k \subseteq \mathcal{N}$ indicates the subset of nodes that belong to NO $k \in \mathcal{K}$. Also, $\mathcal{N}_m \subseteq \mathcal{N}$ indicates the subset of nodes that belong to region $m \in \mathcal{M}$. It is assumed that each NO has installed one cache per region, and that all the caches in a region are interconnected. We also denote with $\lambda_{nvq} > 0$ the expected demand of the users associated to node $n \in \mathcal{N}$ for video $v \in \mathcal{V}$ with required quality $q \in \mathcal{Q}$.

Ideally, each user would like to receive all the layers of the requested video from the local node n since this induces a practically zero delay. If a layer cannot be found at the local cache node n, then n can download it from another node n' in the same region that has already cached it. We denote with $d_{nn'}$ the per unit data delay incurred for this transfer, where it trivially holds that $d_{nn} = 0$, $\forall n \in \mathcal{N}$. As a last resort for node n, the content server can deliver the layer with delay $d_n > d_{nn'}$, $\forall n, n'$. Clearly, a user may download the required layers from different caches or servers. In this case, the user experienced delay will be equal to the *maximum* of the respective delays.

The global caching policy of the cooperating NOs is given by the vector ($x = x_{nvl} \in \{0,1\}, \forall n \in \mathcal{N}, v \in \mathcal{V}, l \in \mathcal{L}$). The objective is to minimize the total delay for satisfying the entire set of requests. The total delay can be written as $J_T^c(x) = \sum_{k \in \mathcal{K}} J_k^c(x)$, where:

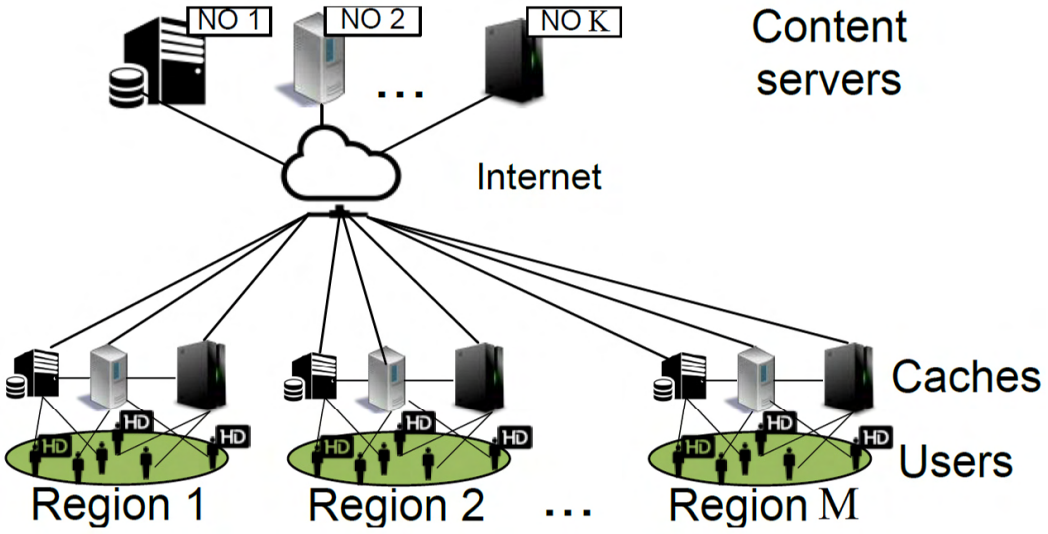


FIGURE 4.3: A distributed caching architecture with K NOs and M regions. Every cache is connected with a distant content server and possibly with other caches in the same region.

$$J_k^c(x) = \sum_{n\in\mathcal{N}_k}\sum_{v\in\mathcal{V}}\sum_{q\in\mathcal{Q}}\lambda_{nvq}\max_{l\in\{1,\ldots,q\}}\left\{\prod_{\substack{n'\in\mathcal{N}:\\ \mathcal{M}_{n'}=\mathcal{M}_n}}(1-x_{n'vl})o_{vl}d_n\right.$$
$$\left.+\Big(1-\prod_{\substack{n'\in\mathcal{N}:\\ \mathcal{M}_{n'}=\mathcal{M}_n}}(1-x_{n'vl})\Big)o_{vl}\min_{\substack{n'\in\mathcal{N}:\\ \mathcal{M}_{n'}=\mathcal{M}_n,\ x_{n'vl}=1}}\{d_{nn'}\}\right\} \tag{4.25}$$

Here, $\mathcal{M}_n \in \mathcal{M}$ indicates the region where node n is located. Every required layer $l \in \{1,...,q\}$ will be delivered to local node n by the content server with per unit data delay d_n if none of the nodes in the same region with n have cached it, i.e., if $\prod_{n'\in\mathcal{N}:\mathcal{M}_{n'}=\mathcal{M}_n}(1-x_{n'vl}) = 1$. Otherwise, among the nodes that have cached l, the one with the lowest delay will deliver it.

4.6.2 Cooperative Caching Policies

The problem of determining the caching policy that *minimizes the user delay of all NOs* can be expressed as follows:

$$\min_{\mathbf{x}} J_T^c(x) \tag{4.26}$$

$$\sum_{v\in\mathcal{V}}\sum_{l\in\mathcal{L}} o_{vl}x_{nvl} \leq S_n, \forall n \in \mathcal{N} \tag{4.27}$$

$$x_{nvl} \in \{0,1\},\ \forall n \in \mathcal{N}, v \in \mathcal{V}, l \in \mathcal{L} \tag{4.28}$$

Decomposition: Since content can only be transferred between nodes in the *same* region, the above problem can be decomposed into M *independent* subproblems, one for each region $m \in \mathcal{M}$. We denote with $\mathcal{N}_m \subseteq \mathcal{N}$ the set of nodes located at region m. For a specific region m, we observe that the total user delay equals to $D_{wc}^m = \sum_{n\in\mathcal{N}_m}\sum_{v\in\mathcal{V}}\sum_{q\in\mathcal{Q}}\lambda_{nvq}o_{v1}d_n$ without using local caching, since all requests are served with layer 1 downloaded by the content servers. Hence, we can express the equivalent problem of *maximizing delay savings for region* m, which we refer to as R_m, as follows:

$$\begin{aligned} R_m: \ \max_{x_m} D_{wc}^m - \sum_{n\in\mathcal{N}_m}\sum_{v\in\mathcal{V}}\sum_{q\in\mathcal{Q}}\lambda_{nvq}\max_{l\in\{1,\ldots,q\}}\Big\{ \\ \prod_{n'\in\mathcal{N}_m}(1-x_{n'vl})o_{vl}d_n + (1-\prod_{n'\in\mathcal{N}_m}(1-x_{n'vl}))o_{vl}d_{nn^*}\Big\} \\ s.t. \sum_{v\in\mathcal{V}}\sum_{l\in\mathcal{L}} o_{vl}x_{nvl} \leq S_n, \forall n \in \mathcal{N}_m \end{aligned} \tag{4.29}$$

$$x_{nvl} \in \{0,1\},\ \forall n \in \mathcal{N}_m,\ v \in \mathcal{V},\ l \in \mathcal{L} \tag{4.30}$$

where $x_m = (x_{nvl} : n \in \mathcal{N}_m, v \in \mathcal{V}, l \in \mathcal{L})$. Each layer l of a file v will be delivered to n by the content server with per unit data delay d_n if none of the nodes have cached it, i.e., if $\prod_{n'\in\mathcal{N}_m}(1-x_{n'vl}) = 1$. Otherwise, among the nodes that have cached l, the one with the lowest delay will deliver it, i.e., the node $n^* =_{n'\in\mathcal{N}_m:x_{n'vl}=1}\{d_{nn'}\}$.

Solution: R_m is a very challenging problem, since each node should seek the best tradeoff between caching the layers of the videos that are popular for its own users (*optimizing local demand*), and caching the ones that are frequently requested by users of other nodes in the same region (*optimizing global demand*). Subsequently, we present an algorithm that achieves an approximation ratio for this important problem. The algorithm partitions the cache space of each node based on an *input parameter* $F \in [0,1]$. Here, F stands for the portion of each cache that is filled in with globally popular video content, while the rest $1-F$ portion is filled in with locally popular video content. Clearly, if $F = 0$, then each node n caches the locally popular video layers independently from the others, while when $F = 1$ all nodes put together their caches and they fill in the union cache space with globally popular video layers.

The proposed algorithm uses the solution to the following type of knapsack problems [87]:

Definition 4.2. *Multiple-Choice Knapsack (MCK)*: Given R classes E_1, E_2,...,E_R of items to pack in a knapsack of capacity W, where item $i \in E_r$ has value p_{ri} and weight w_{ri}, choose *at most one* item from each class such that the total value is maximized without the total weight exceeding W.

Although MCK problem is NP-hard, there exists a *pseudopolynomial-time optimal* algorithm and a *fully-polynomial-time approximation* (FPTA) algorithm to solve it [87]. Pseudopolynomial means that the time is polynomial in the input (knapsack capacity and item weights), but exponential in the length of it (number of digits required to represent it). The FPTA algorithm finds a solution with a performance that is provable no less than $(1-\epsilon)$ times the optimal, while its running time is polynomial to $\frac{1}{\epsilon}$, $\epsilon \in (0,1)$. Therefore, the FPTA algorithm complexity and performance are adjustable, which makes it preferable compared to the first algorithm for large problem instances.

Apart from MCK, the proposed algorithm also uses the solution to the independent caching problem for a single cache-node n, in which case node n is excluded from exchanging content with other nodes. Hence, node n only optimizes its local demand. We denote with P_n this independent caching problem. Then, the following lemma holds.

Lemma 4.3. *The problem P_n is polynomial-time reducible to the MCK problem.*

The proof of this lemma is deferred to Appendix C. The proposed algorithm uses as components the solution to the following two problems:

1. MCK(m): The instance of the MCK problem comprising a knapsack of capacity $F \cdot \sum_{n \in \mathcal{N}_m} S_n$ and V classes of items, each with Q items. The i^{th} item of the v^{th} class has weight $\sum_{l=1}^{i} o_{il}$ and value $\sum_{n \in \mathcal{N}_m} \sum_{q \in \mathcal{Q}} \lambda_{nvq} d_n \sum_{l=1}^{q} (o_{vl} - o_{vl+1}) \prod_{j=1}^{l} (1_{\{j \in \{1,2,\ldots,i\}\}})$, where $1_{\{.\}}$ is the indicator function, i.e. $1_{\{c\}} = 1$ if condition c is true; otherwise it is zero, and $o_{vl+1} = 0$ for $l = q$. Here, the i^{th} item of the v^{th} class corresponds to the first i layers of video v.

2. $P_n(\mathcal{A}_n)$: The instance of the P_n problem in which the layers in the set $\mathcal{A}_n$ are already placed in cache n.

We now present the proposed Layer-aware Cooperative Caching (LCC) algorithm:

- **Stage 1:** Solve the $MCK(m)$ problem. For each item picked in the knapsack, place the corresponding set of layers into the node $n \in \mathcal{N}_m$ with the highest local demand for the respective video. Ensure at each step that at most $F \cdot S_n + s$ amount of data is placed at each node n, where s is the maximum size of an item.
- **Stage 2:** For each node $n \in \mathcal{N}_m$, fill in its remaining cache space by solving the $P_n(\mathcal{A}_n)$ problem, where $\mathcal{A}_n$ consists of the layers placed at n in stage 1.

Theorem 4.4 summarizes one of the main contributions of this section:

Theorem 4.4. *LCC algorithm achieves an approximation ratio of* $\min\{\ \rho\mu,\ \rho'\mu'\ \}$ *for the* R_m *problem, where:*

$$\rho = F - \frac{s}{\sum\limits_{n\in\mathcal{N}_m} S_n},\quad \mu = \min_{n\in\mathcal{N}_m} \frac{\min\limits_{n'\in\mathcal{N}_m\setminus n}\{d_n - d_{nn'}\}}{\max\limits_{n'\in\mathcal{N}_m\setminus n}\{d_n - d_{nn'}\}}$$

$$\rho' = 1 - F - \frac{2s}{\min\limits_{n\in\mathcal{N}_m} S_n},\quad \mu' = \min_{n\in\mathcal{N}_m} \frac{\min\limits_{n'\in\mathcal{N}_m\setminus n} d_{nn'}}{\max\limits_{n'\in\mathcal{N}_m\setminus n} d_{nn'}}$$

The proof of Theorem 4.4 is deferred to Appendix D. The tightness of the approximation ratio of LCC algorithm depends on the delay coefficients $(d_n, d_{nn'},\ \forall n, n' \in \mathcal{N}_m)$, the cache sizes $(C_n, \forall n \in \mathcal{N}_m)$ and the input value F. In the *symmetric case* where $d_n = d$ and $d_{nn'} = d'$, $\forall n, n' \in \mathcal{N}_m$ it becomes: $\mu = 1$ and $\mu' = 1$. When additionally the caches are relatively large, i.e., $\frac{s}{\min_{n\in\mathcal{N}_m} S_n} \to 0$, setting $F = 0.5$ yields an approximation ratio of 0.5, i.e., LCC algorithm achieves at least *half* of the optimal performance.

We note that F is passed as an input to LCC algorithm. A reasonable choice for F is the value that yields the best possible approximation ratio. This requires solving the following optimization problem:

$$\max_{0\le F\le 1} \quad \min\{\ \rho\mu,\ \rho'\mu'\ \} \tag{4.31}$$

Here, the objective function is pointwise minimum of finite number of affine functions and therefore it is concave. Hence, this problem can be solved using standard convex optimization techniques [82].

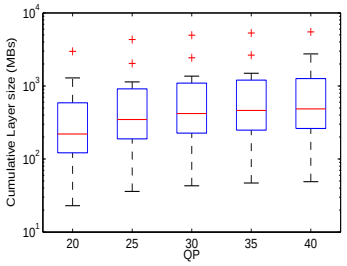


FIGURE 4.4: The cumulative size of the layers required at each quality level for the videos in the library [85]. Each video is encoded into 5 quality levels corresponding to different quantization parameters; $QP \in \{20, 25, 30, 35, 40\}$.

4.6.3 Evaluating Cooperative Caching

In this subsection, we present the numerical results of the experiments that we have conducted to show the superiority of the proposed approach over a commonly used caching scheme. Specifically, we implement the following three algorithms:

- *Independent Caching (IC):* Each NO serves only its own subscribers. For each cache-node n, the caching is performed independently from the rest, by solving the P_n problem.
- *Layer-aware Cooperative Caching (LCC):* The proposed cooperative algorithm according to which all nodes dedicate F fraction of their cache space for storing layers of videos that are globally popular. The remaining space is filled in based on the local video demand.
- *Femtocaching [33]*: This cooperative caching algorithm starts with all the caches being empty. Iteratively, it performs the placement of a layer to a cache that achieves the maximum performance improvement, in terms of total delay (J_T^c). The procedure terminates when there does not exist any cache space available to store content.

The evaluation is carried out for $K = 3$ NOs and a single geographical region. Each NO has installed a cache of capacity equal to S (bytes). The rate of the link between a content server and each of the caches is $1/d_n = 1$mbps, while between any pair of caches it is $1/d_{nn'}$. As a canonical scenario we set $1/d_{nn'} = 5$mbps, while our evaluation also covers the cases where: $1/d_{nn'} \in \{1, 2, ..., 10\}$ mbps.

Requests for a library of $V = 1,000$ popular videos are randomly generated by the users that are associated to the caches. Each video is realized in $Q = 5$ quality levels using SVC. We set the sizes of the $5,000$ respective layers randomly using the real-world

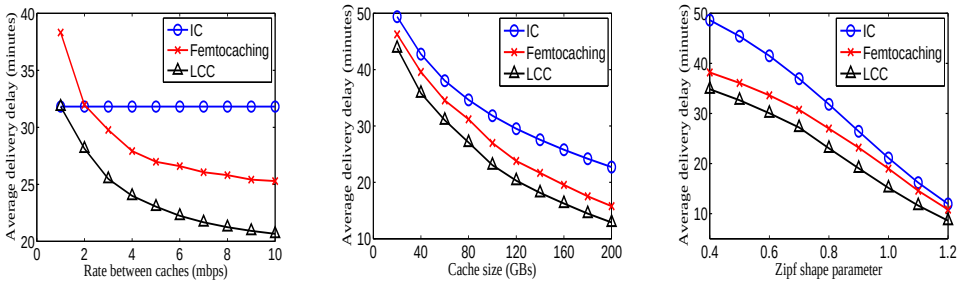
trace in [85]. This dataset contains detailed information about 19 SVC-encoded popular movies spanning 5 SNR quality levels (Figure 4.4). We believe that this is representative of a realistic video delivery system, since layer sizes span two orders of magnitude, and videos of various source formats and publish times are included.

Following empirical studies in VoD systems, we spread the user requests across videos using a Zipf distribution, i.e., the request rate for the i^{th} most popular video is proportional to i^{-z}, for some shape parameter $z > 0$ [61]. We further spread the requests across the $Q = 5$ quality levels uniformly at random. Unless otherwise specified, we set: $S = 100$GBs and $z = 0.8$, while we run the LCC algorithm for each value of F at 0.1 granularity, and pick the value with the lowest total delay.

Impact of rate between caches: We first explore the impact of varying the bandwidth rate between the caches on the average video delivery delay. In the experiment in Figure 4.4(a), the rate spans a wide range of values, starting from 1 to 10 mbps, reflecting different operating conditions. We note that the performance of the IC algorithm is unaffected by this variation, since the caches are excluded from transmitting content one another. On other hand, increasing the rate between caches reduces delay for the cooperative caching algorithms (Femtocaching and LCC), since the layers can be exchanged faster between the caches. *The proposed algorithm (LCC) performs better than the rest for all the rate values.* The delay gains are up to 54% and 22% when compared to IC and Femtocaching algorithm respectively.

Impact of cache sizes: We analyze the impact of cache sizes on algorithms' performance in Figure 4.4(b). As expected, increasing cache sizes reduces delay for all the algorithms as more requests are satisfied locally (without the participation of the content server). IC results in the largest delay compared to the rest schemes (up to 76% difference), since the latter schemes allow the exchange of content between the caches. *The proposed LCC algorithm consistently outperforms Femtocaching, with the gains increasing with cache sizes (up to* 22%*).*

Impact of demand steepness: Finally, we show the impact of the Zipf shape parameter z on algorithms' performance in Figure 4.4(c). As the z value increases the demand distribution becomes steeper and a few videos attract most of the demand. On other hand, a small z value corresponds to an almost uniform demand distribution. We observe that the delay decreases with z for all the algorithms, reflecting that *caching effectiveness improves with the steepness of demand distribution.* LCC performs significantly better than IC and Femtocaching, especially for large z values, with gains up to 39% and 25% respectively.



(a) Impact of rate between caches. (b) Impact of cache sizes. (c) Impact of demand steepness.

Fig. 4.4. (a) The average video delivery delay achieved by IC, Femtocaching and LCC algorithms as a function of (a) the rate between caches, (b) the cache sizes, and (c) the shape parameter of the Zipf distribution.

Main takeaways: For the purpose of optimizing video delivery, the caching problem needs to be revisited so as to additionally take into consideration the different video encoding techniques used to realize different levels of video quality. Layered video encoding technologies, e.g., SVC, offer network flexibility since the layers of each video can be cached at different base stations and/or routed over different paths to users. The benefits mainly depend on the diversity of user demand in terms of the required quality level of video. Cooperative schemes that allow the exchange of content between the caches owned by different network operators can reduce further the video delivery delay.

Chapter 5

Multicast-aware Caching

Contents

5.1 Introduction

Today many operators take advantage of *multicast* to efficiently utilize the available bandwidth of their networks in delivering the same content to multiple receivers [88]. For example, multicast is often used for delivering sponsored content, e.g., mobile advertisements in certain locations, downloading news, stock market reports, weather and sports updates [89]. Meanwhile, multicast has been incorporated in 3GPP specifications in which the proposed technology for LTE is called Evolved Multimedia Broadcast and Multicast Services (eMBMS) [90]. A commercial example of eMBMS is Ericsson LTE Broadcast solution [91]. This technology can be used across multiple cells where

the transmission across them is synchronous using a common carrier frequency. Hence, multicast consumes a subset of the radio resources needed by a unicast service. The remaining resources can be used to support transmissions towards other users, thus enhancing network capacity.

Current proposals from academia and industry consider caching and multicast independently one from the other and for different purposes. On the one hand, caching is used to shift traffic from peak to off-peak hours by exploiting the periodic pattern of traffic generation. This is realized by filling the caches with content during off-peak hours (e.g., nighttime), and serving requests for the stored content by the caches during peak-time (e.g., daytime). On the other hand, multicast is used to reduce energy and bandwidth consumption by serving concurrent user requests for the same content via a single point-to-multipoint transmission instead of many point-to-point (unicast) transmissions. Intuitively, caching should be effective when there is enough *content reuse*; i.e., many recurring requests for a few content files appear over time. Multicast should be effective when there is significant *concurrency* in accessing information across users; i.e., many users concurrently generate requests for the same content file. Such scenarios are more common during crowded events with a large number of co-located people that are interested in the same contents, e.g., during sporting games, concerts and public demonstrations with often tens of thousand attendees [92], [93]. In next generation 5G systems where the demand for mobile data is often massive, and a variety of new services such as social networking platforms and news services employ the one-to-many communication paradigm, e.g., updates in Tweeter, Facebook, etc, it is expected that multicast will be more often applied.

Clearly, it is of paramount importance to design caching and multicast mechanisms for servicing the mobile user requests with the minimum possible energy expenditures. The caching problem differs when multicast is employed to serve concurrent requests for the same content file. Compared to unicast communication, multicast incurs less traffic as the requested file is transmitted to users only once, rather than with many point-to-point transmissions. Hence, the caching problem needs to be revisited to effectively tackle the following questions: *Can caching and multicast be combined to reduce energy costs of an operator? If yes, what is the condition and where the gains come from? From which network parameters or user demand characteristics is this affected?*

In order to answer the above questions, we consider a HCN model that supports caching and multicast for the service of the mobile users. Requests for the same content file generated during a short-time window are aggregated and served through a single multicast transmission when the corresponding window expires (*batching multicast* [94]). To ensure that the user experienced delay will be limited, the duration of this window should

be as small as possible. For example, users may tolerate a very small start-up delay for video streaming applications, whereas larger delay may be acceptable for downloading news, stock market reports, weather and sports updates. The multicast stream can be delivered either by a SBS that is in communication range with the requesters in case that the respective file is available in its cache, or by the MBS which has access to the entire file library through a backhaul link. Clearly, a MBS multicast transmission can satisfy requests generated within the coverage areas of *different* SBSs that have not cached the requested file. However, it typically induces higher energy cost than a SBS, since the distance to the receiver is larger and it also needs to fetch the file via its backhaul link.

First, we demonstrate through simple examples how multicast affects the efficiency of caching policies. Then, we introduce a general optimization problem (which we name MACP) for devising the multicast-aware caching policy that minimizes the overall energy cost. We formally prove the intractability of the MACP problem by reducing it to the set packing problem [95], which is NP-Hard. Following that, we develop an algorithm with performance guarantees under the assumption that the capacity of the caches can be expanded by a bounded factor. This algorithm applies linear relaxation and randomized rounding techniques. Then, we describe a simple heuristic solution that can achieve significant performance gains over existing caching schemes.

Using traffic information from a crowded event with over fifty thousand attendees [92], we investigate numerically the impact of various system parameters, such as the delay tolerance of user application, the SBS cache sizes, the base station transmission cost values and the demand intensity and steepness. We find that the superiority of multicast-aware caching over traditional caching schemes is highly pronounced when: (i) the user demand for content is high and (ii) the user requests for content are delay-tolerant. The gains are 17.5% when users tolerate delay of three minutes, increasing further with the steepness of content access pattern.

Our main technical contributions are as follows:

- *Multicast-aware caching problem (MACP).* We propose a novel caching paradigm and an optimization framework building on the combination of caching and multicast techniques in HCNs. This is important, as content delivery via multicast is part of 3GPP standards and gains increasing interest.
- *Complexity Analysis.* We prove the intractability of the MACP problem by reducing it to the set packing problem [95]. That is, we show that MACP is NP-Hard even to approximate within a factor of $O(\sqrt{N})$, where N is the number of SBSs in a macro-cell. This result reveals how the consideration of multicast transmissions further perplexes the caching problem.

- *Solution algorithms.* Using randomized rounding techniques, we develop a multicast-aware caching algorithm that achieves performance guarantees under the assumption that the capacity constraints can be violated in a bounded way. Also, we describe a simple-to-implement heuristic algorithm that provides significant performance gains compared to the existing caching schemes.

- *Performance Evaluation.* Using system parameters driven from real traffic observations in a crowded event, we show the cases where the next generation HCN systems should optimize caching with concerns on multicast delivery. The proposed algorithms yield significant energy savings over existing caching schemes, which are more pronounced when the demand is massive and the user requests can be delayed by three minutes or more.

The rest of the chapter is organized as follows: Section 5.2 describes the system model and defines the MACP problem formally. In Section 5.3, we show the intractability of the problem and present algorithms with performance guarantees and heuristics. Section 5.4 presents our trace-driven numerical results.

5.2 System Model and Problem Formulation

In this section we introduce the system model, we provide a motivating example that highlights how multicast affects the efficiency of caching policies and, finally, we formally define the multicast-aware caching optimization problem.

5.2.1 System Model

We study the downlink operation of a heterogeneous cellular network (HCN) like the one depicted in Fig. 5.1. A set $\mathcal{N}$ of N small-cell base stations (SBSs) are deployed within a macro-cell,[1] serving the requests of the nearby users. Each SBS $n \in \mathcal{N}$ is equipped with a cache of size $S_n \geq 0$ bytes. The macro-cell base station (MBS) is connected to the core network via a backhaul link through which it receives data from the content servers. We denote with $c_B \geq 0$ the expected energy cost per byte incurred when transferring data via the MBS backahul link. This captures the energy consumed at the aggregation switches, which, based on recent measurement studies [96], increases (approximately) linearly to the total traffic. The backhaul links of the SBSs are usually of low-capacity, e.g., often facilitated by the consumers' home networks such as Digital Subscriber Line

[1]The model can be directly extended for the scenario of more macro-cells and more layers of base stations, e.g., including micro-BSs, femto-cells, etc, where different cells employ different multicasts or coordinate via single-frequency network configurations [90].

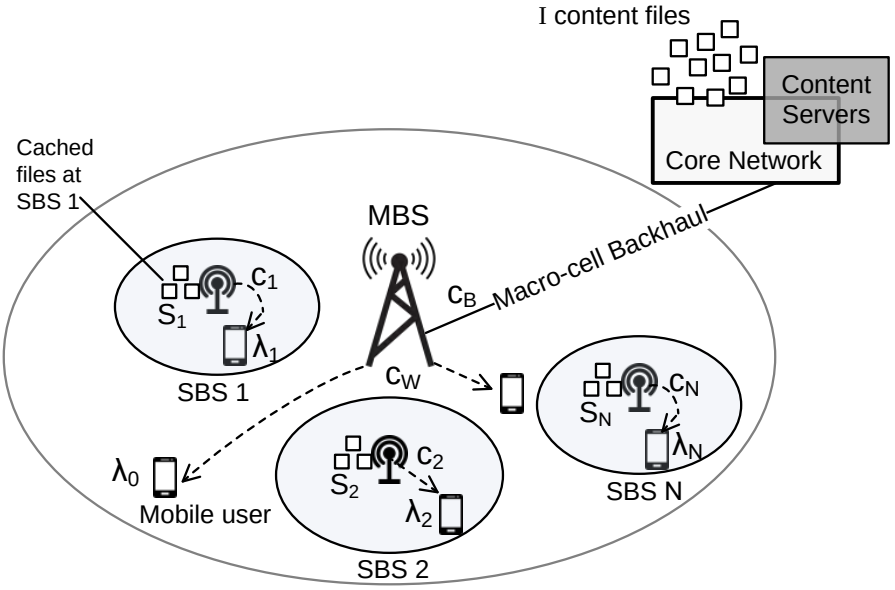


FIGURE 5.1: Graphical illustration of the discussed model. The circles represent the coverage areas of the MBS and the SBSs.

(DSL) [97], and when the content demand is massive they are mainly used to refresh the cached content during off-peak time periods [33]. Therefore, the respective traffic cost can be safely ignored.

Moving to the radio access network, we denote with $c_W \geq 0$ the per byte energy cost incurred when transmitting data directly from MBS to the users in the cell. Although, in many cases energy costs grow super-linearly with wireless traffic [98], recent energy measurements indicated that they can be well approximated by a linear function of the base station load [99]. Similarly, we denote with $c_n \geq 0$ the per byte energy cost incurred when transmitting data from the SBS n to its nearby users. Since the SBSs are in closer proximity to the users than the MBS, it should be: $c_n \leq c_W$, $\forall n \in \mathcal{N}$. We elaborate on the base station transmission cost values in Section 5.4.

The user demand for a set of popular files and within a certain time period is assumed to be known in advance, as in [33]-[35], [18]. Let $\mathcal{I}$ indicate that collection of files, with $I = |\mathcal{I}|$. For notational convenience, we consider all files to have the same size normalized to 1. To facilitate the analysis, we consider the case that the coverage areas of the SBSs are non-overlapping and denote with $\lambda_{ni} \geq 0$ the average demand for file i generated by the users located in the coverage area of SBS n.[2] Also, $\lambda_{0i} \geq 0$ denotes the average demand for file i generated by users who are not in the coverage area of any of the SBSs[3].

[2]The model can be directly extended to handle the case of overlapping SBS coverage areas.

[3]Notice that the current practice of operators is to deploy SBSs to certain areas with high traffic. Hence, other less congested areas may be covered only by the MBS.

The operator employs multicast (such as eMBMS) for transmission of the same content to multiple receivers. In this case, user requests within a short-time window are aggregated and served through a single multicast stream when the corresponding window expires. We denote with d (time units) the time duration of this window, also called *multicast period*. We then denote with p_{ni} the probability that *at least one* request for file i is generated by users in the coverage area[4] of SBS n (area n) during a multicast period. Similarly, p_{0i} indicates the respective probability for the users that are not in the coverage area of any of the SBSs (area n_0). The collection of all subsets of areas excluding the empty set can be defined as follows:

$$\mathcal{R} = (r : r \subseteq \mathcal{N} \cup n_0, r \neq \emptyset). \tag{5.1}$$

Then, we denote with q_{ri} the probability that at least one request for the file $i \in \mathcal{I}$ is generated within each one of the areas $r \in \mathcal{R}$, during a multicast period. For example, if requests are generated *independently* among different areas, then the following equation holds:

$$q_{ri} = \prod_{n \in r}(p_{ni}) \cdot \prod_{n \notin r}(1 - p_{ni}). \tag{5.2}$$

Our model is generic, since it allows for any probability distribution.

We consider the more general case in which both the SBSs and the MBS can use multicast. Namely, a multicast transmission of SBS $n \in \mathcal{N}$ satisfies the requests for a cached file generated by users within its coverage area, while a MBS transmission satisfies requests generated within the coverage areas of different SBSs (and requests from area n_0) that have not cached the requested file. This latter option induces higher cost since the MBS has higher transmission cost ($c_W \geq c_n$) and also needs to fetch the file via its backhaul link ($c_B > 0$).[5] This exactly is the main contribution of this work: *"To carefully design the caching policy with concerns on the multicast transmissions so as to minimize the overall energy cost"*.

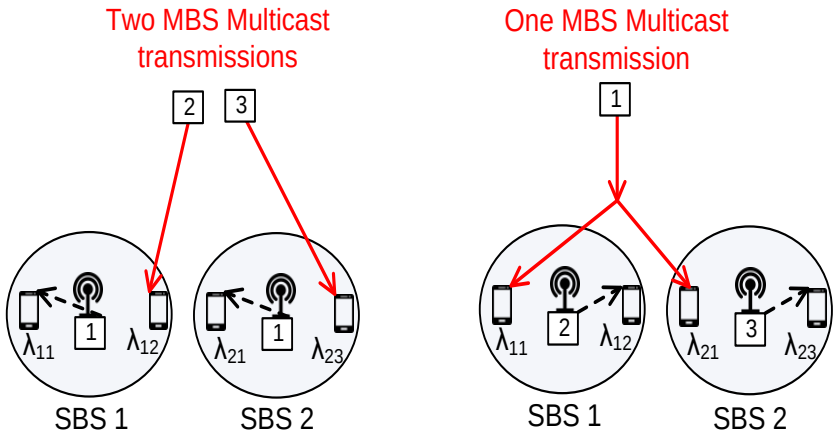
Before we introduce formally the problem, let us provide a simple example that highlights how the consideration of multicast transmissions perplexes the caching problem.

5.2.2 Motivating Example

Let us consider a multicast service system with two SBSs ($\mathcal{N} = \{1, 2\}$) and three files ($\mathcal{I} = \{1, 2, 3\}$). Each SBS can cache at most one file because of its limited cache size.

[4] With a slight abuse of notation we use the same index for base stations and their coverage area.

[5] Notice that our model can be extended for the case that the MBS is also equipped with a cache, where the locally cached files are directly delivered to the users by the MBS without using the backhaul link.



(a) Conventional caching. (b) Multicast-aware caching.

FIGURE 5.2: An example with two SBSs and three files when (a) conventional and (b) multicast-aware caching is applied. The labels below SBSs represent the cached files. The labels on the top represent the files delivered by MBS.

We set $c_B + c_W = 1$, $c_1 = c_2 = 0$ and $d = 1$. We also set the number of requests for each file i within the coverage area of a SBS n to follow a Poisson probability distribution with rate parameter λ_{ni}, $\forall n \in \mathcal{N}, i \in \mathcal{I}$. Hence, the probability that at least one request for file i is generated within SBS n in the time period d is:

$$p_{ni} = 1 - e^{-\lambda_{ni} d} \tag{5.3}$$

Finally, we set $\lambda_{11} = 0.51$, $\lambda_{12} = 0.49$, $\lambda_{13} = 0$, $\lambda_{21} = 0.51$, $\lambda_{22} = 0$, and $\lambda_{23} = 0.49$, which imply that $p_{11} = 0.3995$, $p_{12} = 0.3874$, $p_{13} = 0$, $p_{21} = 0.3995$, $p_{22} = 0$ and $p_{23} = 0.3874$.

In a conventional system, each user request is served via a point-to-point unicast transmission. It is well known that placing the most popular files with respect to the local demand in each cache is optimal (in terms of the overall energy cost) in this setting. Hence, the optimal caching policy places file 1, which is the most popular file, to both SBS caches. By applying the above caching policy to the multicast service system that we consider here, all the requests for file 1 will be satisfied by the accessed SBSs at zero cost. The requests within SBS 1 for file 2 and the requests within SBS 2 for file 3 will be served by the MBS at per unit cost $c_B + c_W$ (Fig. 5.2(a)). Assuming independent generation of requests, the total energy cost will be: $(c_B + c_W) \cdot p_{12} \cdot (1 - p_{23}) + (c_B + c_W) \cdot (1 - p_{12}) \cdot p_{23} + 2 \cdot (c_B + c_W) \cdot p_{12} \cdot p_{23} = 0.7747$, where the last term in the summation is multiplied by 2 because two *different* files are requested for download and thus two MBS transmissions are required for serving the requests.

However, if we take into consideration the fact that the user requests are aggregated and served via multicast transmissions every $d = 1$ time unit, then *the optimal caching policy changes*; it places file 2 to SBS 1 and file 3 to SBS 2. In this case, all the requests for file 1 will be served by the MBS via a single multicast transmission of cost $c_B + c_W$ (Fig. 5.2(b)). The requests for the rest files will be satisfied by the accessed SBSs at zero cost. Hence, the total energy cost will be: $(c_B+c_W)\cdot\big(p_{11}\cdot(1-p_{21})+(1-p_{11})\cdot p_{21}+p_{11}\cdot p_{21}\big) = 0.6394 < 0.7747$.

This example demonstrated the inefficiency of conventional caching schemes that neglect multicast transmissions when determining the file placement to the caches. Novel schemes are needed that combine caching with multicast to better exploit the available cache space.

5.2.3 Problem Formulation

Let us introduce the binary optimization variable x_{ni} that indicates whether file $i \in \mathcal{I}$ is placed at the cache of SBS $n \in \mathcal{N}$ ($x_{ni} = 1$) or not ($x_{ni} = 0$) [6]. These variables constitute the *caching policy* of the operator:

$$x = (x_{ni} \in \{0, 1\} : n \in \mathcal{N}, i \in \mathcal{I}) \tag{5.4}$$

We also consider a subset $r \in \mathcal{R}$ that includes the areas receiving requests for a file $i \in \mathcal{I}$. In this case, a MBS multicast transmission of file i will occur if a requester cannot find i in an accessed SBS. This implies that *at least one* of the following conditions holds: (i) a request for file i is generated within an area that is not in the coverage area of any of the SBSs, i.e., $n_0 \in r$, or (ii) a request for file i is generated within the coverage area of a SBS $n \in r \setminus n_0$, but the latter has not stored in its cache the requested file. We use the term y_{ri} to indicate whether a MBS multicast transmission will occur ($y_{ri} = 1$) or not ($y_{ri} = 0$). Based on the above discussion, y_{ri} is a function of the caching policy x:

$$y_{ri} = \max\Big\{ \max_{n\in r\setminus n_0} \{1 - x_{ni}\}, 1_{\{n_0\in r\}} \Big\} \tag{5.5}$$

where $1_{\{.\}}$ is the indicator function, i.e., $1_{\{b\}} = 1$ iff condition b is true; otherwise $1_{\{b\}} = 0$. The external max term is equal to 1 if at least one of the two internal terms is equal to 1.

Let us now denote with $J_i(x)$ the energy cost for servicing all the requests for a file i, which clearly depends on the caching policy x. For each subset of areas r that generate

[6] Note that we do not consider fragmentation of files and placement of file fractions at different caches. Instead, we tackle the (more challenging) integer-nature caching problem in this work.

requests for file i within a time period, a single MBS multicast transmission of cost $c_B + c_W$ occurs, if a requester cannot find i in an accessed SBS ($y_{ri} = 1$). In other case ($y_{ri} = 0$), all the requests are satisfied by the accessed SBSs, where the requests in area n incur cost c_n. Hence:

$$J_i(x) = \sum_{r \in \mathcal{R}} q_{ri} \cdot \left(y_{ri} \cdot (c_B + c_W) + (1 - y_{ri}) \cdot \sum_{n \in r} c_n \right) \tag{5.6}$$

The *Multicast-Aware Caching Problem* (MACP) determines the caching policy that minimizes the total (across all files) energy cost[7]:

$$\text{MACP: minimize}_x \sum_{i \in \mathcal{I}} J_i(x) \tag{5.7}$$

$$\text{subject to: } \sum_{i \in \mathcal{I}} x_{ni} \leq S_n, \ \forall n \in \mathcal{N} \tag{5.8}$$

$$x_{ni} \in \{0, 1\}, \ \forall n \in \mathcal{N}, \ i \in \mathcal{I}, \tag{5.9}$$

where inequalities in (5.8) ensure that the total amount of data placed at each cache will not exceed its capacity. Constraints in (5.9) indicate the discrete nature of the optimization variables.

MACP is an integer non-linear programming problem and hence it is challenging to solve. Also, its objective function in (5.7) has an exponentially long description in the number of SBSs N, since the summation in $J_i(x)$ is over all subsets $r \in \mathcal{R}$. As we formally prove in the next section, MACP is an NP-Hard problem.

5.3 Complexity and Solution Algorithms

In this section, we prove the high complexity of the MACP problem and present solution algorithms with performance guarantees and heuristics.

5.3.1 Complexity

In this subsection, we prove that the MACP problem cannot be approximated within any ratio better than the square root of the number of SBSs. The proof is based on

[7] We emphasize that our model is focused on the energy consumed for transmitting data to users. Hence, other factors such as cooling [99] and caching energy costs [100] are left outside the scope of our study.

a reduction from the well known NP-Hard set packing problem (SPP) [95]. In other words, we prove that SPP is a special case of MACP. Particularly, the following theorem holds:

Theorem 5.1. It is NP-Hard to approximate MACP within any ratio better than $O(\sqrt{N})$.

Theorem 5.1 is of high importance, since it reveals how the consideration of multicast transmissions further perplexes the caching problem. In order to prove Theorem 5.1 we will consider the corresponding (and equivalent) decision problem, called Multicast-Aware Caching Decision Problem (MACDP). Specifically:

MACDP: Given a set $\mathcal{N}$ of SBSs, a set $\mathcal{I}$ of files, the cache sizes S_n $\forall n \in \mathcal{N}$, the costs c_B, c_W and c_n $\forall n \in \mathcal{N}$, the multicast period d, the probabilities q_{ri} $\forall r \in \mathcal{R}, i \in \mathcal{I}$, and a real number $Q \geq 0$, *we ask the following question*: does there exist a caching policy x, such that the value of the objective function in (5.7) is less or equal to Q and constraints (5.8)-(5.9) are satisfied?

The set packing decision problem is defined as follows:

SPP: Consider a finite set of elements $\mathcal{E}$ and a list $\mathcal{L}$ containing subsets of $\mathcal{E}$. We ask: do there exist k subsets in $\mathcal{L}$ that are pairwise disjoint?

Lemma 5.2. *SPP problem is polynomial-time reducible to the MACDP.*

Proof. Consider the *SPP* decision problem and a specific instance of MACDP with $N = |\mathcal{E}|$ SBSs, i.e., $\mathcal{N} = \{1, 2, \ldots, |\mathcal{E}|\}$, $I = |\mathcal{L}|$ files, i.e., $\mathcal{I} = \{1, 2, \ldots, |\mathcal{L}|\}$, unit-sized caches: $S_n = 1$ $\forall n \in \mathcal{N}$, $c_B + c_W = 1$, and $c_n = 0$ $\forall n \in \mathcal{N}$. Parameter d is any positive number, and the question is if we can satisfy all the user requests with energy cost $Q = 1 - \frac{k}{|\mathcal{L}|}$, where k is the parameter from the SPP. The important point is that we define the q_{ri} elements as follows:

$$q_{ri} = \begin{cases} 1/|\mathcal{L}|, & \text{if } r = \mathcal{L}(i) \\ 0, & \text{else} \end{cases} \tag{5.10}$$

where $\mathcal{L}(i)$ is the i^{th} component of the list $\mathcal{L}$. Notice that with the previous definitions, $\mathcal{L}(i)$ contains a certain subset of elements of $\mathcal{E}$. For the MACDP, under the above mapping, this corresponds to a subset of SBSs asking with a non-zero probability file i. Moreover, with (5.10) we assume that these probabilities are equal for all files $i \in \mathcal{I}$ and have value $1/|\mathcal{L}|$.

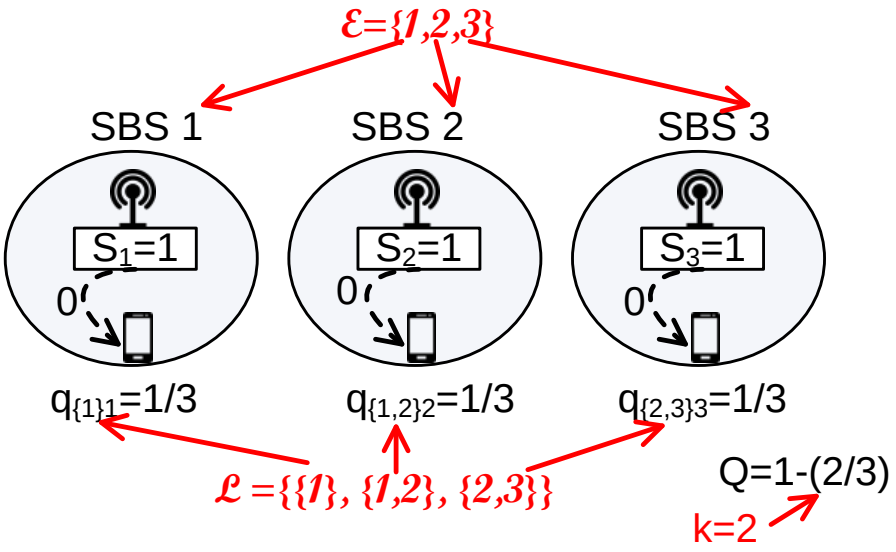


FIGURE 5.3: An example of the reduction from SPP with $\mathcal{E} = \{1,2,3\}$, $\mathcal{L} = \{\{1\},\{1,2\},\{2,3\}\}$ and $k = 2$. In the MACDP instance there are $N = |\mathcal{E}| = 3$ SBSs and $I = |\mathcal{L}| = 3$ files. There is a solution to MACDP of cost $Q = 1 - \frac{2}{3}$ that places file 1 to SBS 1 and file 3 to SBSs 2 and 3. Accordingly, the solution to SPP picks $k = 2$ subsets: $\mathcal{L}(1) = \{1\}$ and $\mathcal{L}(3) = \{2,3\}$.

If the MBS serves all the requests, then the MACDP problem has a value (cost) of $c_B + c_W = 1$ (the worst case scenario). For each file i that the operator manages to serve completely through local caching at the SBSs, the operator reduces its cost by $(c_B + c_W) \cdot q_{ri} = 1/|\mathcal{L}|$. This reduction is ensured only if the file is cached in all the SBSs $n \in r$ for which $q_{ri} = 1/|\mathcal{L}|$. Therefore, in order to achieve the desirable value $Q = 1 - \frac{k}{|\mathcal{L}|}$, we need to serve locally the requests for k files. That is, to find k subsets of SBSs where the file requested by these SBSs will be cached (so as to avoid MBS multicasts).

Notice that each cache can store up to one file. Hence, the caching decisions should be *disjoint* with respect to the SBSs. For example, in Fig. 5.3, SBS 1 cannot store both files 1 and 2, because $S_1 = 1$. This ensures that the subsets $\{1\}$ and $\{1,2\}$ in the SPP problem will not be both selected. In other words, the value of the objective function in (5.7) can be less or equal to $1 - \frac{k}{|\mathcal{L}|}$, if there exist k subsets in $\mathcal{L}$ that are pairwise disjoint.

Conversely, if a Set Packing for some k exists, then for each subset $\mathcal{L}(i)$ that is picked in it, one can place the file i to the cache of each one of the SBSs $n \in \mathcal{L}(i)$ corresponding to this subset. At most one file is placed in each cache, since the selected subsets in the list are pairwise disjoint. The cost will be equal to $1 - \frac{k}{|\mathcal{L}|}$. □

SPP is NP-Hard and moreover it is inapproximable within $O(\sqrt{|\mathcal{E}|})$ [95]. According to the reduction, we create a SBS for each one of the elements in $\mathcal{E}$, and hence it holds $|\mathcal{E}| = N$, which completes the proof of Theorem 5.1.

5.3.2 Algorithm with performance guarantees

In this subsection, we present a caching algorithm with performance guarantees. We first note that, based on Theorem 5.1, it is unlikely to find a tight approximate solution to the MACP problem. Hence, we follow an alternative approach by letting the solution to violate the cache capacity constraints in equation (5.8) by a bounded factor. Such a constraint violation turns out to greatly facilitate the solution of the problem. Following that, we present a provably near-optimal solution algorithm applying linear relaxation and randomized rounding techniques, variants of which have been also used for optimizing graph cuts [103].

To start with, we express the y_{ri} terms, introduced in equation (5.5), as binary optimization variables and denote with y the respective vector:

$$y = (y_{ri} \in \{0,1\} : r \in \mathcal{R}, i \in \mathcal{I}) \tag{5.11}$$

Then, the equivalent to the MACP problem, which we refer to as MACP$'$, that optimizes *both* the values in x and y can be expressed as follows:

$$\text{MACP}'\text{: minimize}_{x,y} \sum_{i \in \mathcal{I}} J_i(x,y) \tag{5.12}$$

$$\text{subject to: constraints: (5.8), (5.9)}$$

$$y_{ri} \geq 1 - x_{ni},\ \forall r \in \mathcal{R},\ i \in \mathcal{I},\ n \in r \tag{5.13}$$

$$y_{ri} \geq 1_{\{n_0 \in r\}},\ \forall r \in \mathcal{R},\ i \in \mathcal{I} \tag{5.14}$$

$$y_{ri} \in \{0,1\},\ \forall r \in \mathcal{R}, i \in \mathcal{I} \tag{5.15}$$

where $J_i(x,y)$, is given by equation (5.6), and it is now explicitly expressed as a function of both the x and y variables. Constraints (5.13)-(5.14) stem of decomposition of equation (5.5), dictating that y_{ri} is equal to 1 if either there is a SBS in r that has not cached file i or $n_0 \in r$.

Then, we introduce the *linear relaxation* of the MACP$'$ problem, which we refer to as LR(MACP$'$). This differs from MACP$'$ in that the variables in x and y can take any real value within $[0,1]$, i.e., constraints (5.9) and (5.15) are replaced by $x_{ni} \in [0,1],\ \forall n \in \mathcal{N}, i \in \mathcal{I}$ and $y_{ri} \in [0,1],\ \forall r \in \mathcal{R}, i \in \mathcal{I}$. The objective function and the constraints of the LR(MACP$'$) problem are linear with respect to the optimization variables. Hence, it

Algorithm 5.1: Randomized rounding algorithm with parameter $\mu \in (0, \frac{1}{2})$

```
Let (x†, y†) be the optimal solution to LR(MACP′);
Choose m ∈ [1/2 − μ, 1/2 + μ] uniformly at random;
Let A = {(r, i) : r ∈ R, i ∈ I, y†_ri ≥ m};
Let B = {(r, i) : r ∈ R, i ∈ I, y†_ri < m};
Set y_ri = 1 ∀(r, i) ∈ A, and y_ri = 0 ∀(r, i) ∈ B;
for n ∈ N, i ∈ I do
    if ∃ r : y_ri = 0 and n ∈ r then
        x_ni ← 1;
    else
        x_ni ← 0;
Output x;
```

can be solved using standard linear optimization techniques [56]. We need to emphasize at this point that the number of optimization variables in the LR(MACP′) problem is non-polynomial to the number of SBSs N, since there is a variable for each subset $r \in \mathcal{R}$ (equation (6.4)). In practice though, the number of SBSs in a macro-cell is small (e.g., a few tens), and hence we can apply software toolboxes like CPLEX and Mosek [57] to efficiently solve LR(MACP′).

Having found a fractional solution to the LR(MACP′) problem, denoted with $(x^\dagger, y^\dagger)$, the proposed algorithm applies randomized rounding techniques to approximate the (integer) solution of the MACP problem. Specifically, given an input parameter value $\mu \in (0, \frac{1}{2})$, the algorithm decides uniformly at random a threshold value $m \in [\frac{1}{2}-\mu, \frac{1}{2}+\mu]$. Then, iteratively it rounds each y_{ri} variable to 1 if its (fractional) value exceeds m (subset $\mathcal{A}$); otherwise it takes the 0 value (subset $\mathcal{B}$). Finally, a variable x_{ni} will take the value 1, if there exists y_{ri} variable with $n \in r$ that was rounded to 0; otherwise it takes the 0 value. The procedure is summarized in Algorithm 5.1.

Theorem 5.3. *Algorithm 5.1 outputs a caching policy of energy cost at most* $\frac{2}{1-2\mu}$ *times the optimal. The expected amount of data placed in each cache is at most* $\frac{1}{2\mu}$ *times its capacity.*

Proof. Let V_{opt} and V_1 indicate the optimal solution value for the MACP problem and the one achieved by Algorithm 5.1 respectively. Then, it holds that:

$$
\begin{aligned}
V_{\text{opt}} &\geq \sum_{r\in\mathcal{R}}\sum_{i\in\mathcal{I}} q_{ri}\Big(y^{\dagger}_{ri}(c_B+c_W)+(1-y^{\dagger}_{ri})\sum_{n\in\mathcal{N}} c_n\Big)\\
&\geq \sum_{(r,i)\in\mathcal{A}} q_{ri}y^{\dagger}_{ri}(c_B+c_W)+\sum_{(r,i)\in\mathcal{B}} q_{ri}(1-y^{\dagger}_{ri})\sum_{n\in\mathcal{N}} c_n\\
&\geq \sum_{(r,i)\in\mathcal{A}} q_{ri}(\frac{1}{2}-\mu)(c_B+c_W)+\sum_{(r,i)\in\mathcal{B}} q_{ri}(\frac{1}{2}-\mu)\sum_{n\in\mathcal{N}} c_n\\
&= (\frac{1}{2}-\mu)V_1, \qquad (5.16)
\end{aligned}
$$

where the first inequality is because the optimal solution of the linear relaxed problem provides a lower bound to the optimal solution value of the initial problem. The second inequality is because we kept in the summation only a subset of the terms and all the terms are positive, i.e., $q_{ri}\geq 0$, $y^{\dagger}_{ri}\geq 0$, $1-y^{\dagger}_{ri}\geq 0$, $c_B+c_W\geq 0$, $c_n\geq 0$. The third inequality is because: $y^{\dagger}_{ri}\geq m\geq \frac{1}{2}-\mu$, $\forall(r,i)\in\mathcal{A}$ and $y^{\dagger}_{ri}<m\leq\frac{1}{2}+\mu$, $\forall(r,i)\in\mathcal{B}$.

We also note that the m value is picked uniformly at random from an interval of size 2μ. According to step 7 of Algorithm 5.1, a file i will be placed at a SBS cache n ($x_{ni}=1$) only if there exists $r\in\mathcal{R}$ for which $n\in r$ and $y_{ri}=0$. Variable y_{ri} takes the zero value when m is larger than $y^{\dagger}_{ri}$, which happens with probability at most $\frac{1-y^{\dagger}_{ri}}{2\mu}$. Hence, the probability that x_{ni} takes the value 1 is at most:

$$
\frac{1-\min\limits_{r\in\mathcal{R}:n\in r} y^{\dagger}_{ri}}{2\mu} \overset{(5.13)}{\leq} \frac{x^{\dagger}_{ni}}{2\mu} \qquad (5.17)
$$

Summing over all the files yields that the expected amount of data placed in a SBS cache $n\in\mathcal{N}$ is at most:

$$
\sum_{i\in\mathcal{I}}\big(\frac{x^{\dagger}_{ni}}{2\mu}\big) \overset{(5.8)}{\leq} \frac{1}{2\mu}\cdot S_n \qquad (5.18)
$$

□

For example, picking the value $\mu=\frac{1}{6}$ will result a solution of cost that is at most three times larger than the optimal violating cache capacities by a factor less than three. Picking a lower value μ yields a tighter performance guarantee, but increases the factor within which the cache capacities are violated. Hence, the parameter value μ can be used to control the trade off between performance and robustness of the solution, where different operators may decide different μ values based on their priorities.

Constructing a feasible solution. We note that, as the cache capacities of the SBSs may be violated by a factor of $\frac{1}{2\mu}$ when applying Algorithm 5.1, the operator may not be able to store and deliver through the SBSs all the files required to ensure the performance

guarantee of our algorithm ($\frac{2}{1-2\mu}$). In this case, an option for the operator is to expand the cache capacities by a factor of $\frac{1}{2\mu}$. Nevertheless, the operator is often unwilling (or, incapable) to perform additional investments. Hence, it is needed to convert the solution of Algorithm 5.1 into a feasible solution, i.e., a solution that satisfies equation (5.8).

To obtain such a solution, we start with the solution obtained by Algorithm 5.1. Then, iteratively, we perform the removal of a file to a SBS cache that yields the minimum cost increment (with respect to the objective in (5.7)). At each iteration, we ensure that the SBSs with remaining amount of cached data that is lower or equal to their capacities are excluded from content removal. The procedure terminates when there does not exist any SBS available to remove content.

Please notice that, the above conversion may deteriorate the quality of the solution of Algorithm 5.1. Unfortunately, we cannot derive a tight theoretical performance bound for the obtained solution due to hardness of the MACP problem (as we described in Theorem 5.1). However, as we show with an extensive numerical study in the next section, the obtained solution operates very close to the optimal one in realistic settings.

5.3.3 Heuristic algorithm

Finally, we present an alternate algorithm which, in contrast to the previous algorithm, finds a solution to the MACP problem in a greedy manner, rather than using a systematic optimization procedure.

The proposed iterative algorithm starts with all the caches being empty. At each iteration, it places the file to a non-full cache that yields the lowest value to the objective function in (5.7). The iteration terminates when all the caches become full. This is a greedy ascending procedure that can be summarized in Algorithm 5.2.

Specifically, I_n is the number of files already placed at the cache of SBS n at every iteration of the algorithm, and $(\times)$ denotes the cartesian product of two sets. The set $\mathcal{D}$ includes all the pairs (n, i) for which the placement of file i at the cache of SBS n has not been performed yet, and the cache of n has not been filled up yet. Let $f(x, n, i)$ be the value of the objective function of the MACP for the file placement x, where we additionally set $x_{ni} = 1$. At every iteration, Algorithm 5.2 picks the pair $(n^*, i^*) \in \mathcal{D}$ with the lowest cost value $f(x, n^*, i^*)$. This corresponds to the placement of the file i^* at the cache of the SBS n^*. If the cache of SBS n^* becomes full, then the algorithm excludes all the pairs (n^*, i), $\forall i$, from the set $\mathcal{D}$. This way, no more files will be placed at this cache. The algorithm terminates when all the caches become full.

Algorithm 5.2: Heuristic algorithm

$x \leftarrow [0, ..., 0]$;
$I_n \leftarrow 0, \forall n \in \mathcal{N}$;
$\mathcal{D} \leftarrow \mathcal{N} \times \mathcal{I}$;
for $t = 1, 2, ..., \sum_{n \in \mathcal{N}}(S_n)$ **do**
    $(n^*, i^*) \leftarrow \text{argmin}_{(n,i) \in \mathcal{D}} f(x, n, i)$;
    $x_{n^* i^*} = 1$;
    $\mathcal{D} \leftarrow \mathcal{D} \setminus (n^*, i^*)$;
    $I_{n^*} \leftarrow I_{n^*} + 1$;
    if $I_{n^*} = S_{n^*}$ **then**
        for $i \in \mathcal{I}$ *such that* $(n^*, i) \in \mathcal{D}$ **do**
            $\mathcal{D} \leftarrow \mathcal{D} \setminus (n^*, i)$
Output x;

Algorithm 5.2 requires $\sum_{n=1}^{N}(S_n)$ iterations to terminate. At each iteration it evaluates the value of the objective function after each one of $N \cdot I$ candidate file placements. We need to emphasize at this point that a similar technique has been shown to achieve an approximation ratio of 2 for the conventional caching problem of unicast transmissions (without multicast), where a different objective function was optimized [33]. However, the problem we consider in this chapter is of higher complexity, as we showed in Theorem 5.1. Despite the lack of any theoretical performance guarantees, Algorithm 5.2 provides significant performance gains over existing caching schemes in practical scenarios, as we will show numerically in the next section.

5.4 Performance Evaluation

In this section, we numerically evaluate the energy savings achieved by the proposed multicast-aware caching algorithms over existing caching strategies. The main part of the evaluation is carried out for a sporting event with thousand attendees [92] covered by a macrocell and several SBSs. Additional scenarios differing in the population density, number of SBSs and energy costs are evaluated, which lead to an understanding of how the savings vary in different regions and markets. Overall, we find that moving from a conventional caching scheme to one enhanced with multicast-awareness can indeed reduce energy costs, and the benefits are higher when the demand is massive and the user requests for content are delay-tolerant. For a crowded event, an operator can improve its bottom line by 17.5% by delivering multicast streams every 3 minutes, with the gains increasing further with the steepness of content access pattern. In the rest of this section, we discuss these results in detail; we begin by describing the algorithms and the simulation setup used in the later evaluations.

5.4.1 Algorithms and evaluation setup

Throughout, we compare the performance of five schemes:

1. *Popularity-Aware Caching & Unicast Transmissions (PAC-UT)*: The standard mode of operation currently in use in many caching systems. Each SBS stores in its cache the locally most popular files independently from the others. Each user request is served by a separate unicast transmission.

2. *Popularity-Aware Caching & Multicast Transmissions (PAC-MT)*: Similar to PAC-UT, differing in that requests for the same file within the same multicast period are served by a single multicast transmission.

3. *Linear-Relaxed Multicast-Aware Caching & Multicast Transmissions (LMAC-MT)*: We apply Algorithm 1 with $\mu = 1/6$ to decide the cache placement. The placement is further processed to yield a feasible solution as described in the end of Subsection 5.3.2. User requests for the same file within the same multicast period are served by a single multicast transmission.

4. *Greedy Multicast-Aware Caching & Multicast Transmissions (GMAC-MT)*: Similar to LMAC-MT, differing in that we apply Algorithm 2 to decide the cache placement.

5. *Lower-bound (LB)*: The lower bound to the optimal solution of the MACP problem found by solving the linear relaxed problem (LR(MACP$'$)). Since, this solution is not feasible, it is only used as a benchmark for measuring the efficacy of the proposed algorithms.

We need to emphasize that, in order to solve the linear problem in LMAC-MT and LB schemes, we executed code from the Visual Studio environment using the Mosek Optimization Toolbox [57]. This software uses interior point methods and sparse arrays to store variables, which, for our setup, yield a running time in the scale of minutes.

The main part of the evaluation is carried out for a sporting event with macrocell coverage and stadium-wide deployment of $N = 14$ SBSs as in Figure 5.4. The system parameters are set using the measured trace of content requests collected during the 2013 Superbowl in February at the New Orleans Superdome [92]. During this event, over fifty thousand users generated around three thousand requests for a set of $I = 1,000$ popular files. Considering that all requests appear during the four-hour period of the game, this results to an average rate of ≈ 12.5 requests per minute. To model the user demand in our evaluation, we uniformly spread the requests in the trace across

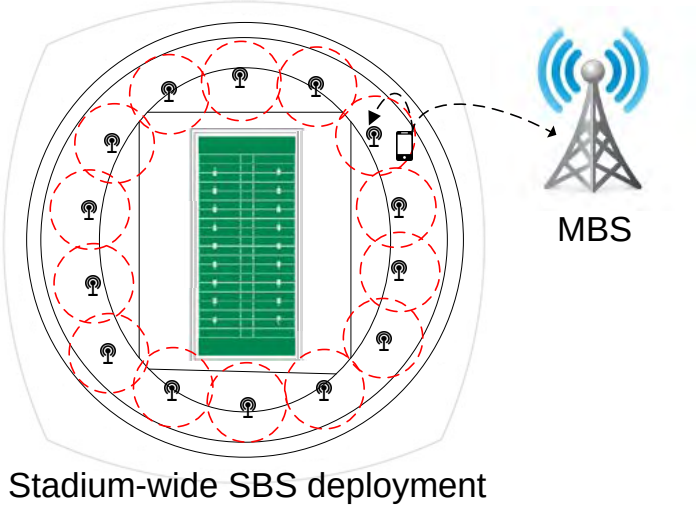


FIGURE 5.4: A stadium-wide deployment of SBSs. The dashed circles represent the coverage areas of the SBSs. A user can be served either by the neighbor SBS or by the collocated MBS.

the coverage regions of the 14 SBSs. We further spread these requests across files using a Zipf popularity distribution with shape parameter z [61]. This results the demand values λ_{ni} for each SBS n and file i. We also set $\lambda_{0i} = 0$, $\forall i \in \mathcal{I}$. For the computation of p_{ni} and q_{ri} probabilities, we assume that request generation follows an independent Poisson distribution (Eq. (5.2) and Eq. (5.3)). Unless otherwise specified, all file sizes are normalized to 1 and each SBS is equipped with a cache that can store up to 20% of the entire file library size. Finally, we set $z = 1.2$ and $d = 3$ minutes, while our evaluation also covers a wide range of z and d values.

Following recent measurement traces in 3G networks, we approximate MBS power consumption by a linear function of the carried traffic load, with the slope being $8.25/G_{MBS}$, where G_{MBS} denotes the bandwidth capacity of the MBS (cf. Figure 3 in [99]). Since, the MBS capacities are typically dimensioned based on the anticipated demand, we set G_{MBS} to be capable of handling all the user requests in our simulation, i.e., $G_{MBS} = 12.5$ (requests per minute); therefore it is $c_W = 8.25/12.5$. The MBS backhaul power includes the power consumed at the aggregation switches, which increases linearly to the total traffic, with the slope being $(1-\alpha)\frac{Ag_{switch}}{Ag_{max}}P_{max}$ [96]. Here, P_{max} represents the maximum power consumption of the switch, Ag_{switch} is the amount of carried traffic, Ag_{max} is the maximum amount of traffic a switch can handle and $\alpha \in (0,1)$. We set $P_{max} = 300$ (Watts), $Ag_{max} = 24 \cdot G_{MBS}$ and $\alpha = 0.1$ (as in Table II in [96]); therefore it is $c_B = 30/(24 \cdot 12.5)$. SBS energy consumption is typically lower than the one for the MBS, due to the closer proximity to the users, with the actual value depending on the type of the SBS and its coverage (e.g. pico-cell, femto-cell). As a canonical scenario we

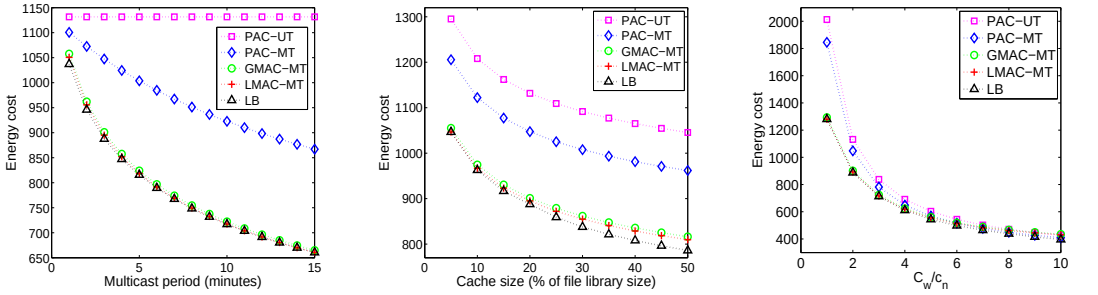
set $c_n = c_W/2$, while our evaluation also covers the cases where: $\frac{c_W}{c_n} \in \{1, 2, ..., 10\}$ [96].

5.4.2 Evaluation results

We compare the energy cost achieved by the above schemes as a function of the duration of multicast period, the cache sizes and the base station transmission costs. Following that, we investigate how the population density, the intensity and steepness of demand and the number of SBSs impact the results.

Impact of the duration of the multicast period: Intuitively, multicast will be effective when there is significant concurrency in accessing content across users, i.e., many requests for the same file frequently appear within a multicast period. In this case, the requests are aggregated and served via a common multicast stream (instead of many unicast transmissions) improving the energy efficiency of the system. Although, this may occasionally be the case for typical urban macrocells with a few hundred or thousand users, our analysis reveals that it is particularly relevant during crowded events with tens of thousand people collocated in the same area. For the specific sporting event that we consider in the evaluation, Figure 5.5(a) shows the energy cost achieved by the discussed schemes when the duration of the multicast period d is varied within 1 to 15 minutes. We observe that the performance gap between each one of the schemes that enable mulitcast transmissions (PAC-MT, LMAC-MT, GMAC-MT and LB) and the PAC-UT increases with d. This was expected, since increasing d increases the probability of receiving multicast requests for a file within a period. Importantly, the proposed multicast-aware caching schemes (LMAC-MT, GMAC-MT) consistently outperform PAC-MT, with the gains increasing with d (up to 31%). Even for a relatively small value of d, multicast-aware caching schemes achieve significant energy savings over the conventional caching scheme. For example, the gains are 17.5% for $d = 3$. This is of high importance since users are unlikely to tolerate large delays in receiving content. Interestingly, the proposed schemes operate very close to LB and hence the optimal solution (less than 2% worse).

Impact of cache sizes: We analyze the impact of the cache sizes on the algorithms' performance in Figure 5.5(b). Here, the cache size of each SBS is varied from 5% to 50% of the entire file library size. As expected, increasing cache sizes reduces energy costs for all schemes as more requests are satisfied locally (without the participation of the MBS). PAC-UT results in the largest energy cost compared to the rest schemes (up to 30% difference), since the latter schemes serve many aggregated requests via a single multicast instead of many unicast transmissions. The proposed multicat-aware caching schemes (LMAC-MT and GMAC-MT) consistently outperform the popularity-aware caching scheme PAC-MT, with the gains increasing with cache sizes (up to 19%).



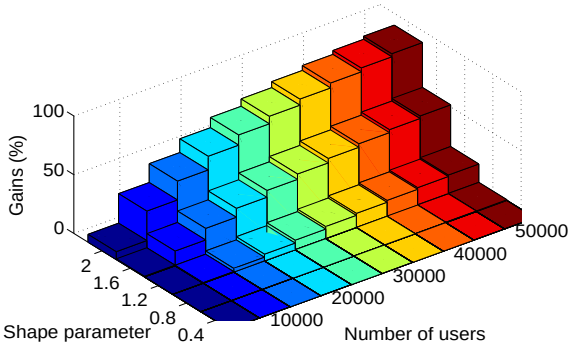
(a) Impact of multicast period (b) Impact of cache sizes (c) Impact of transmission costs

FIGURE 5.5: Energy cost achieved by PAC-UT, PAC-MT, LMAC-MT, GMAC-MT and LB schemes for various values of: (a) the multicast time period, (b) the cache size of each SBS and (c) the base station transmission costs.

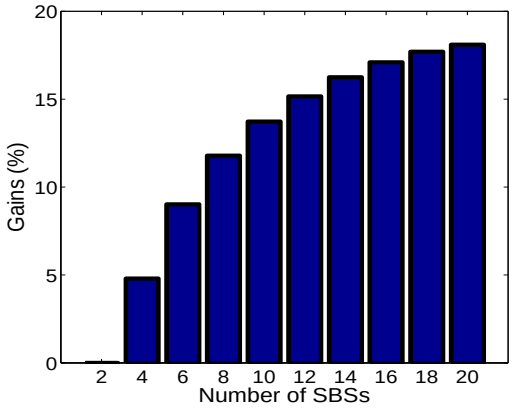
More importantly, LMAC-MT and GMAC-MT operate very close to LB -and hence the optimal solution- for all the cache sizes (less than 4% worse).

Impact of base station transmission costs: We explore the impact of the the base station transmission cost parameters on the algorithms' performance in Figure 5.5(c). Particularly, we keep c_W constant and alter the c_n values within $\{c_W/1, c_W/2, ..., c_W/10\}$. We observe that as the ratio c_W/c_n increases, the energy cost achieved by all the schemes decreases since the cost incurred by the service at the SBSs becomes lower. The popularity-aware caching schemes (PAC-UT and PAC-MT) are the most sensitive to this alteration. Again, LMAC-MT and GMAC-MT outperform the popularity-aware schemes, especially for low values of c_W/c_n. For $c_W = c_n$, the gains are 57% and 44% when compared to the PAC-UT and PAC-MT scheme respectively. However, when c_n values become relatively low compared to c_W, the performance of the PAC-MT scheme comes very close to the multicast-aware schemes. This is because, the file popularity distribution is the same across all the SBSs (homogeneous demand) in our experiment, and hence simply replicating the (same) most popular files at all the caches drastically reduces the number of multicast-transmissions employed by the MBS.

Impact of demand patterns and number of SBSs: The demand patterns used in Figures 5.5(a)-5.5(c) may seem contrived, but in fact, they are very much in line with recent traffic measurements reported during crowded events [92]. To obtain a holistic view of the benefits of enhancing the caching scheme with multicast-awareness we repeat the experiments for different values of intensity and steepness of demand. Specifically, we consider ten scenarios with five to fifty thousand users generating requests for files. The intensity of demand for the case of fifty thousand people matches the one used for the sporting game in Figures 5.5(a)-5.5(c). For the rest choices, the demand intensity is scaled down proportionally to the number of users. For each scenario, five different choices of the Zipf shape parameter z are evaluated. Here, $z = 0.4$ indicates an almost



(a) Impact of user demand



(b) Impact of number of SBSs

FIGURE 5.6: Gains of multicast-aware caching (GMAC-MT) over conventional caching (PAC-MT) as a function of (a) the intensity and steepness of demand for content and (b) the number of SBSs.

uniform content popularity distribution, whereas $z = 2$ stands for a high-steep distribution. The 3-D barplot in Figure 5.6(a) shows that the energy gains of a multicast-aware caching scheme (GMAC-MT) over a conventional caching scheme (PAC-MT) increase as either the intensity or the steepness of demand increases. In the best scenario, with fifty thousand users and $z = 2$, the gains are more than 90%.

Finally, we explore how the number of SBSs N impacts the results. The barplot in Figure 5.6(b) shows that the energy gains of a multicast-aware caching scheme (GMAC-MT) over a conventional caching scheme (PAC-MT) increase as N increases. For example, the gains grow from 4.8% when $N = 4$ to 15.2% when $N = 12$, and further increase to 18.1% for $N = 20$. This is because, increasing N makes it more likely that concurrent requests for the same file occur at different SBSs, which implies a higher number of MBS multicast transmissions. This in turn calls for a careful cache-design that intelligently balances the number of requests served via MBS and SBS multicast.

Main takeaways: Caching can be combined with multicast to reduce the energy expenses required for serving the mobile users. Taking into consideration multicast when designing the caching policy perplexes the problem further. A simple iterative as well as a randomized-greedy caching algorithm can yield significant energy savings over existing caching schemes, which are more pronounced when the demand is massive and the user requests can be delayed by three minutes or more.

Chapter 6

Mobility-aware Caching

Contents

6.1 Introduction

In the emerging hyper-dense deployments of Small-Cell Base Stations (SBSs) [104], [105] mobile users may be frequently handed off between SBSs. This transition can take place

within a few minutes considering the typical range of SBS coverage areas [106]. For a user who is repeatedly moving in and out of the SBS coverage areas, the system can deliver *parts* of the requested file through different SBS caches that the user encounters. If certain file parts fail to be delivered on time by the SBSs, the request is redirected to the macro-cell network. The latter option raises scalability concerns, especially during peak usage hours.

Clearly, mobility affects the efficiency of the caching policies. For example, on a several blocks long shopping road, users may frequently move in and out of the coverage areas of two SBSs located a few blocks away one from another. These users see a distributed cache that is the union of the two SBS caches. Therefore, there is no benefit from replicating the same file parts at both the SBSs. Disjoint file parts should be cached instead. Clearly, the caching policy should be designed with concerns on predictions about user mobility patterns. Such a consideration adds up to the complexity of the traditional caching problem where optimization is based solely on the anticipated content demand [33], [35], [8].

In this chapter, we revisit the caching problem taking into consideration the mobility of the users. We first show that the problem of deriving the caching policy that maximizes the requests served by the caches is NP-Hard to approximate within any constant factor. Then, as a first attempt to overcome this difficulty, we present a *Mixed Integer Programming* formulation to find optimal centralized solutions using branch and bound techniques [110].

Following the intuition that the user's future position highly depends on the current one, we model user movements via *random walks on a Markov chain* [107]. Going one step further, we assign weights to the states of the chain representing the amount of "useful" data that a user can download by the SBS caches at each time instance. Here, by "useful" data, we refer to the parts of the requested file which were not previously downloaded by the user. Therefore, the total weight of a walk determines whether the request will be redirected to the MBS or not, and the framework enables the operator to minimize the load of the macro-cell network. Using large deviation inequalities specific to the Markov model, we derive a *distributed caching algorithm* that leverages mobility predictions to minimize the probability that a request reaches the macro-cell network. To better utilize the cache space, our scheme applies *Maximum Distance Separable (MDS) codes* [108], [109] to store at the SBSs encoded versions of the files instead of the raw data packets. In this sense, a file request is completely served when the total amount of (encoded) data downloaded by the user is at least equal to the size of the requested file. This facilitates analysis and can potentially increase the efficiency of content access compared to the traditional case that uncoded file segments are cached.

Using measured traces of human mobility patterns in wireless network environments and requests for Youtube videos, we investigate numerically the impact of various parameters on the efficiency of the caching decisions, such as the SBS cache sizes, the delay deadline, the density of SBS deployment and the transmission capacities of SBSs. We find that the proposed algorithm can offload up to 65% more traffic of the macro-cells than conventional caching algorithms in realistic settings. The technical contributions of this work can be summarized as follows:

- *Modeling.* We introduce the caching problem in HCNs compromising users moving in and out of the SBS coverage areas with the goal of minimizing the probability that a request reaches the macro-cell network.
- *Complexity.* We prove the caching problem to be NP-Hard to approximate within any constant factor. The proof is based on a reduction from the Independent Set Decision Problem [95].
- *Centralized small-scale solution.* We formulate the caching problem as a Mixed Integer Programming (MIP) problem, and give a centralized solution using branch and bound techniques.
- *Distributed large-scale solution.* We introduce an optimization framework that relates the probability that a request reaches the macro-cell network to the total weight of a random walk in a Markov chain. Using large deviation inequalities, we derive a distributed caching algorithm that scales well with problem size.
- *Performance evaluation.* We use real traces of mobility patterns and requests for Youtube videos and show that our approach can offload up to 65% more traffic of the macro-cells than existing caching algorithms.

The rest of the chapter is organized as follows: Section 6.2 describes the system model and introduces formally the caching problem. In Section 6.3, we prove the intractability of the problem and present a centralized solution that is applicable for small problem sizes. A connection to the Markov chain model and a distributed solution that scales well with problem size are presented in Section 6.4. Finally, Section 6.5 presents our evaluation results.

6.2 System model and problem formulation

In this section, we describe the system model, we present a simple example that shows how mobility affects the efficiency of the caching policies, and we formally define the optimization problem.

6.2.1 System model

We study the downlink operation of a heterogeneous cellular network like the one depicted in Figure 6.1. A set $\mathcal{N}$ of N small-cell base stations (SBSs) are deployed in a macro-cell operating in conjunction with the conventional macro-cell base station (MBS)[1]. Each SBS $n \in \mathcal{N}$ is equipped with a cache of size C_n (bytes). Since the coverage areas of the SBSs may overlap one another, a user may be concurrently covered by multiple SBSs. Mobile users may repeatedly move in and out of the SBS coverage areas, thus associating with different SBSs at different times.

To model user mobility, we introduce a set $\mathcal{L}$ of L highly visited locations in the macro-cell, e.g., busy shopping blocks, hotspots, crowded crossroads, etc. These locations can be extracted using clustering algorithms on the user mobility traces [111]. Each location $l \in \mathcal{L}$ may be covered by multiple SBSs, denoted by $\mathcal{N}_l \subseteq \mathcal{N}$. We then partition time into identical slots and allow users to move to different locations from slot to slot as in [107]. We assume that the operator leverages previous time period statistics to estimate user's location [112], [113]. In this sense, we denote with p_l the probability that a user is in location $l \in \mathcal{L}$. We also denote with $q_{ll'}$ the probability that a user moves from location l to location l', $\forall l, l' \in \mathcal{L}$ within a time slot. Intuitively, the probability $q_{ll'}$ will be higher for locations l and l' that are in close proximity one another.

The average mobile user demand for a set $\mathcal{F}$ of F popular content files and within a certain time period (e.g., a few hours or days) is assumed to be fixed and known in advance, as in [33]. Specifically, for a request generated in location l, we denote with λ_{lf} the probability that f is the requested file. This captures the interest/preferences of the users in location l for content which may vary from location to location. For example, users on a shopping road may be particularly interested in fashion content, while users in proximity to stock market may frequently ask for financial reports. The size of file $f \in \mathcal{F}$ is denoted with $s_f > 0$ (bytes).

We consider *delayed data offloading* and associate each request with a deadline d. That is, each request must be served within a specified time window of d slots by the encountered SBSs, or it will be redirected to the MBS. Clearly, a user visits d locations within the deadline. We refer to the sequence of visited locations as the *walk* of the user, i.e., $w = (w_1, w_2, \ldots, w_d)$, where $w_i \in \mathcal{L}$ denotes the location visited at slot $i \in \{1, 2, \ldots, d\}$. Since a user may visit multiple times the same location, the walk w is a *multiset*, i.e., it possibly includes duplicate elements. Let us denote with $\mathcal{W}$ the set of all possible walks.

[1] Our model can be directly extended for multiple macro-cells and MBSs.

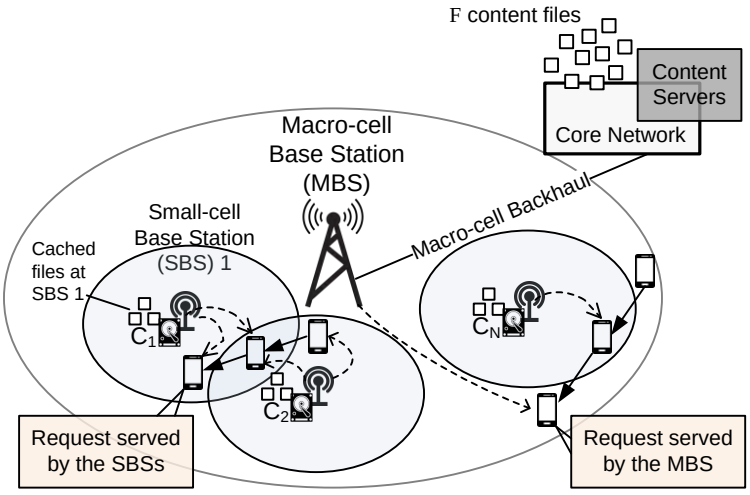


FIGURE 6.1: Graphical illustration of the proposed model. Circles represent the coverage areas of the MBS and the SBSs. Solid and dashed arrows specify user trajectories and base station associations respectively.

Then, the probability that a user takes a walk $w \in \mathcal{W}$ is given by:

$$r_w = p_{w_1} \prod_{i=1}^{d-1} q_{w_i w_{i+1}} \tag{6.1}$$

We consider a massive content delivery scenario, e.g., in populated areas or during peak traffic hours. Hence, the bottleneck in content delivery is not the receiver antenna, but the limited transmission capacity of the SBSs. In this sense, we denote with $B_n \geq 0$ the average amount of data that SBS n can transmit to a user within a time slot. The different B_n values across the SBSs reflect the heterogeneity in terms of the deployed bandwidth and the average workload.

To better utilize the SBS cache space, we adopt a Maximum Distance Separable (MDS) code [108], [109] and store encoded versions of the files instead of the raw data packets. Particularly, a set of encoded segments for each file is generated. Since these segments can be treated equally, we only need to consider how many segments, rather than exactly which segments, of a file to store at each cache. Successful file recovery occurs when the total amount of encoded data is at least equal to the size of the original (uncoded) file.

Our goal is to determine the content placement at each SBS to fully utilize the network resources, such as the limited cache sizes of the SBSs, and the limited contact duration with the users. Traditional caching schemes neglect user mobility and contact duration limits and store complete copies of the files at the caches [33], [35]-[8]. In the following, we show the inefficiency of such schemes using a simple example.

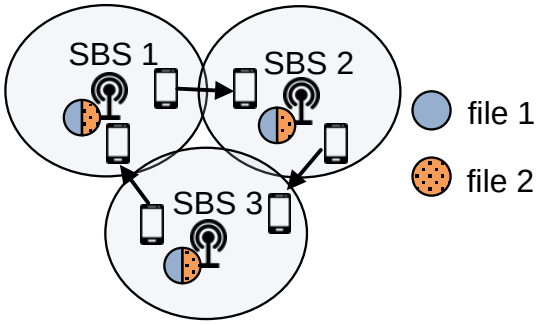


FIGURE 6.2: An example with $N = 3$ SBSs, $F = 2$ unit-sized files and unit-sized caches. Mobile users contact uniformly at random a pair of two SBSs within $d = 2$ slots deadline. During each contact half unit of data can be transmitted.

6.2.2 Motivating example

Let us consider the scenario in Figure 6.2 with $N = 3$ SBSs each one equipped with a unit-sized cache. Within a deadline of $d = 2$ time slots, each mobile user contacts uniformly at random a sequence of two SBSs as he moves. During each contact only half unit of data can be transmitted due to the time slot duration limits. There exist also $F = 2$ equally-popular files each one of size one.

In a mobility-agnostic system, each user is assumed to be stationary and hence he can be served by the local SBS. It is well known that placing the most popular files with respect to the local demand in each cache is optimal (in terms of the macro-cell load) in this setting [33]. Since the two files are equally popular in our example, there is an indifference in caching them by the operator. Hence, the optimal caching policy would store a complete copy of either file 1 or file 2 at each SBS. If all SBSs store the same file, say file 1, then all the users can successfully download this file by the two encountered SBSs; half unit of data by each one of them. However, all the requests for file 2 will be redirected to MBS. In the other case that an SBS, say SBS 1, stores a different file (file 2), then only the users requesting file 1 contacting the two last SBSs would successfully download the requested file within the deadline. The rest users will be redirected to the MBS, since they will download at most half of the requested file by the encountered SBSs.

However, if we take into consideration the fact that the users move and access pairs of SBSs, then the optimal caching policy changes; it stores half unit of MDS-encoded data of each file at each SBS. In this case, no matter which file a user requests, he can download half of it by each encountered SBS. The downloaded encoded parts can be combined to recover the requested (uncoded) file. Hence, none of the requests will

be redirected to MBS. This example, reveals the *inefficiency of conventional caching schemes that neglect user mobility and contact duration limits.*

6.2.3 Problem formulation

We introduce the optimization variable $x_{nf} \in [0,1]$ to indicate the portion of encoded data of file f that is cached at SBS n. These variables constitute the caching policy of the operator:

$$x = (x_{nf} \in [0,1] \ : \ \forall n \in \mathcal{N}, \ f \in \mathcal{F}) \tag{6.2}$$

A user may encounter the same SBS multiple times during his walk. We denote with $\mathcal{N}_w \subseteq \mathcal{N}$ the set of SBSs encountered at least once during the walk w, and α_{wn} the exact number of appearances of SBS n in w. Clearly, it is wasteful for a user to download the same data already downloaded by an SBS at previous contacts. Specifically, during the 1^{st} contact with SBS n, the *useful* portion of file f that can be downloaded is given by:

$$y_{nf}^1 = \min\{x_{nf}, \ \frac{B_n}{s_f}\} \tag{6.3}$$

i.e., it is upper bounded by the portion of cached data and the SBS transmission capacity. During the k^{th} contact with SBS n, where $k \in \{2,3,\ldots,\alpha_{wn}\}$, the *useful* portion of file f is given by:

$$y_{nf}^k = \min\{x_{nf} - \sum_{t=1}^{k-1} y_{nf}^t, \ \frac{B_n}{s_f}\} \tag{6.4}$$

where we have subtracted from x_{nf} the portion of file f downloaded at the $k-1$ previous contacts.

A user request will reach the MBS if the total portion of the requested file that is downloaded by the SBSs is less than 1. To express the probability of this event, we note that, for a user taking walk w and requesting file f, the total portion of the file f downloaded by the encountered SBSs is equal to $\sum_{n\in\mathcal{N}_w}\sum_{k=1}^{\alpha_{wn}} y_{nf}^k$. Since the latter indirectly depends on the caching policy x (cf. equations (6.3), (6.4)), the probability that a user request reaches the MBS can be expressed as follows:

$$J(x) = \sum_{w\in\mathcal{W}} r_w \sum_{f\in\mathcal{F}} \lambda_{w_1 f} 1_{\{\sum_{n\in\mathcal{N}_w}\sum_{k=1}^{\alpha_{wn}} y_{nf}^k < 1\}} \tag{6.5}$$

where $1_{\{.\}}$ is the indicator function, i.e., $1_{\{c\}} = 1$ iff condition c is true; otherwise $1_{\{c\}} = 0$.

The problem of deriving the caching policy that minimizes the probability that a request reaches the MBS can be expressed as follows:

$$\min_{x} \; J(x) \tag{6.6}$$

$$s.t. \; \sum_{f \in \mathcal{F}} s_f x_{nf} \leq C_n, \forall n \in \mathcal{N} \tag{6.7}$$

$$x_{nf} \in [0,1], \forall n \in \mathcal{N}, f \in \mathcal{F} \tag{6.8}$$

where constraints in (6.7) indicate that the total amount of data placed in a cache should not exceed its capacity. Inequalities in (6.8) indicate the non-negativeness of the optimization variables and that it would be wasteful to place to a cache more than one unit of a file. The above problem is difficult to solve due to it's non-convex nature and the high number of different walks that a user can take. Namely, there exist L^d such walks. In the next two sections, we prove the intractability of the problem and provide efficient solutions.

6.3 Complexity and centralized small-scale solution

In this section, we formally prove the intractability of the caching problem and show how to formulate it as a *Mixed Integer Programming* (MIP) problem. This is important since there exist many commercial packages, such as CPLEX [110], that allow for efficient solution to such problems.

6.3.1 Complexity

As Lemma 6.1 states, the caching problem is NP-Hard to approximate within any constant factor.

Lemma 6.1. *It is NP-Hard to approximate the problem described in (6.6)-(6.8) within any constant factor.*

Proof. To prove lemma 6.1, we consider the corresponding decision problem, Caching Decision Problem (CDP): *CDP:* Given the values of the terms d, r_w, $\mathcal{N}_w$, α_{wn}, λ_{lf}, s_f, B_n and C_n and a number $Q > 0$, we ask: does there exist a caching policy x, such that the value of the objective function in (6.6) is less or equal to Q and constraints (6.7)-(6.8) are satisfied?

We will prove the NP-Hardness of CDP by reduction from the independent set decision problem (ISDP) [95]. *ISDP:* Consider an undirected graph with a set $\mathcal{V}$ of V vertices

and a set $\mathcal{E}$ of E edges. We ask: do there exist k vertices that are non-adjacent, i.e., for every two vertices there is no edge connecting the two?

We will show that every instance of the ISDP problem can be expressed as a specific instance of the CDP problem. We construct this instance as follows:

We create V walks, one walk for each vertex $v \in \mathcal{V}$. We denote with $w(v)$ the walk corresponding to vertex v. For each walk, we also create a unit-sized file and a user that requests this file following this walk. Each walk includes a sequence of d SBSs without duplicates, where each SBS corresponds to a separate location. Each SBS can deliver arbitrarily large amount of data per slot, i.e., $B_n = +\infty, \forall n$. We also restrict the aggregate cache space of the SBSs in a walk $w(v)$ to be equal to 1, i.e., $\sum_{n \in \mathcal{N}_{w(v)}} C_n = 1$, $\forall v \in \mathcal{V}$. The important point is that we force every two walks $w(v_1)$ and $w(v_2)$ for which the vertices v_1 and v_2 are adjacent in the ISDP instance to share a common SBS.

If the MBS serves all the requests, then the objective function in (6.6) has a value equal to 1 (the worst case scenario). For each user that the operator manages to serve completely through local caching at the SBSs, the operator reduces this value by $1/V$. Hence, there will be a caching policy with value equal to $1 - k/V$ if there are k users that are served by the SBSs during their walks. Notice that the aggregate amount of data that the SBSs in a walk can cache is at most 1 and each user requests a separate file. Hence, the caching decisions should be disjoint with respect to the k walks. Since every walk corresponds to a separate vertex, this occurs when there are k vertices in the ISDP instance that are non-adjacent. Hence, the CDP instance is equivalent to the ISDP instance.

ISDP is NP-Hard to approximate within any constant factor [95]. Hence, the above reduction indicates the inapproximability of CDP as well and completes the proof of Lemma 6.1. □

6.3.2 MIP formulation

To obtain the MIP formulation of the caching problem, we express the y_{nf}^k terms, introduced in equations (6.3)-(6.4), as optimization variables and denote with y the respective vector:

$$y = (y_{nf}^k \in [0,1] : n \in \mathcal{N}, f \in \mathcal{F}, k \in \{1, 2, \ldots, d\}) \tag{6.9}$$

We also introduce the *integer* optimization variables z:

$$z = (z_{wf} \in \{0,1\} : w \in \mathcal{W}, f \in \mathcal{F}) \tag{6.10}$$

Here, z_{wf} indicates whether file f will be delivered to a user taking a walk w by the MBS ($z_{wf} = 1$) or not ($z_{wf} = 0$).

Then, the MIP problem can be expressed as follows:

$$\min_{x,y,z} \sum_{w \in \mathcal{W}} r_w \sum_{f \in \mathcal{F}} \lambda_{w_1 f} z_{wf} \tag{6.11}$$

s.t. (6.7) – (6.8)

$$y_{nf}^k \in [0, \frac{B_n}{s_f}], \ \forall n \in \mathcal{N}, f \in \mathcal{F}, k = 1, ..., d \tag{6.12}$$

$$\sum_{k=1}^{d} y_{nf}^k \leq x_{nf}, \ \forall n \in \mathcal{N}, f \in \mathcal{F} \tag{6.13}$$

$$z_{wf} \geq 1 - \sum_{n \in \mathcal{N}_w} \sum_{k=1}^{\alpha_{wn}} y_{nf}^k, \ \forall w \in \mathcal{W}, f \in \mathcal{F} \tag{6.14}$$

$$z_{wf} \in \{0, 1\}, \forall w \in \mathcal{W}, f \in \mathcal{F} \tag{6.15}$$

Inequalities (6.12)-(6.13) stem of decomposition of inequalities (6.3)-(6.4). Inequality (6.14) restricts that z_{wf} will be 1 if $\sum_{n \in \mathcal{N}_w} \sum_{k=1}^{\alpha_{wn}} y_{nf}^k < 1$.

In practice, MIP problems can be solved only for small-scale instances, i.e., involving a few number of SBSs, locations and files. This is because the applied Branch & Bound methods perform implicit enumeration of the solution space, partitioning it into a search tree of exponential size. Also, a central entity is required to compute and communicate the solution to the SBSs. In the following section, we will show how to derive an efficient distributed solution for arbitrarily large problem instances.

6.4 Distributed large-scale solution

In this section, we establish a distributed solution for large-scale caching systems. We first show how the caching problem relates to the Markov chain model, then we establish an upper bound on the objective function in (6.6) using large deviation inequalities, and finally we propose a distributed algorithm that minimizes this upper bound.

6.4.1 Relation to the Markov chain model

We introduce a Markov chain with state space $\mathcal{S}$ and matrix M of transition probabilities. A sequence of states $(S_1, S_2, \ldots, S_t)$ indicates a t-step random walk on the chain starting from an initial distribution Φ on $\mathcal{S}$. States can be assigned weights based on a function $\Omega : \mathcal{S} \to [0, 1]$. In this case, the total weight of a walk $(S_1, S_2, \ldots, S_t)$ is equal to

$\sum_{i=1}^{t} \Omega(S_i)$. Given an instance of the caching problem, we construct the corresponding Markov chain as follows.

State space $\mathcal{S}$. The state space consists of the following: (i) a *root* state indexed by 0, and (ii) a group of $\sum_{t=1}^{d} L^t$ *inner* states for each file $f \in \mathcal{F}$. The states of each group are partitioned into d tiers, where tier 1 contains the first L states, tier 2 the next L^2 states and so on. We denote with $\mathcal{F}_u \in \{1, 2, \ldots, F\}$ and $\mathcal{T}_u \in \{1, 2, \ldots, d\}$ the group and the tier of a state u respectively. Here, a state u belonging to tier $t \in \{1, 2, \ldots, d-1\}$ is the unique *parent* of L states in tier $t+1$. We call the latter states as the *children* of state u and denote with $\mathcal{C}_u \subseteq \mathcal{S}$ the respective set. We also introduce the notation $\mathcal{L}_u \in \{1, 2, \ldots, L\}$ for a state u, where $\mathcal{L}_u = l$ if u is the l^{th} child of another inner state. Similarly, the root state is the unique parent of all the states in tier 1, L for each group. For a tier-1 state u that is the l^{th} child of the root in a particular group, we set $\mathcal{L}_u = l$.

Transition matrix M. The probability of transiting from state $u \in \mathcal{S}$ to $v \in \mathcal{S}$ is given by:

$$M_{uv} = \begin{cases} 1-\alpha, & if\ u = v = 0 \\ \alpha \cdot p_{\mathcal{L}_u} \cdot \lambda_{\mathcal{L}_u \mathcal{F}_u}, & if\ u = 0, \mathcal{T}_v = 1 \\ q_{\mathcal{L}_u \mathcal{L}_v}, & \text{if } 0 < \mathcal{T}_u < d, v \in \mathcal{C}_u \\ 1, & \text{if } \mathcal{T}_u = d, v = 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.16}$$

where $a \in (0, 1)$ is any constant. Therefore, during a walk, when at the root state, the chain can either stay in it or move to a tier-1 state at the next step. Then, the chain moves to a tier-2 state and so on. Having reached a tier-d state, the chain moves back to the root. Figure 6.3 illustrates an example for $N = 2$ SBSs, $L = 2$ locations, $F = 2$ files and $d = 3$ slots deadline. Here, each location corresponds to a single SBS coverage area.

Initial Probability Φ. The probability that a random walk starts from a state $u \in \mathcal{S}$ is given by:

$$\Phi(u) = \begin{cases} p_{\mathcal{L}_u} \cdot \lambda_{\mathcal{L}_u \mathcal{F}_u}, & \text{if } \mathcal{T}_u = 1 \\ 0, & \text{otherwise} \end{cases} \tag{6.17}$$

Therefore, every walk starts from a tier-1 state.

We can show that a user's walk in the macro-cell in the caching problem can be expressed as random walk on the above chain. Specifically, each inner state u represents a specific location $\mathcal{L}_u \in \mathcal{L}$ visited by a user requesting file $\mathcal{F}_u \in \mathcal{F}$ at time slot $\mathcal{T}_u \in \{1, 2, \ldots, d\}$.

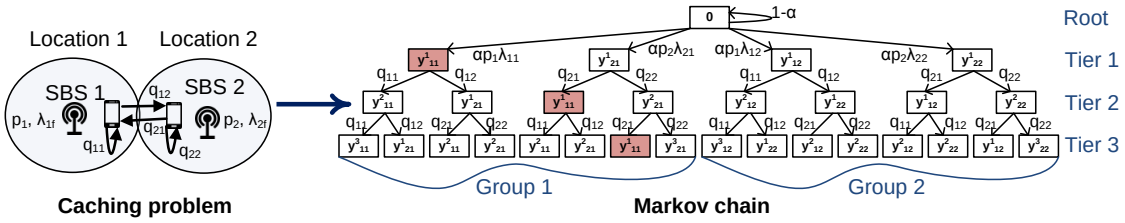


FIGURE 6.3: An example of the Markov chain with $N = 2$ SBSs, $L = 2$ locations, $F = 2$ files and $d = 3$ slots. In the chain, rectangles denote states and state labels indicate the respective weights. Link labels specify the transition probabilities. To ease presentation, the links uniting tier-d states to the root are omitted.

By construction, a random walk starting from a tier-1 state traverses $\mathcal{T}_u - 1$ states before reaching u. Equivalently, the user visits $\mathcal{T}_u - 1$ locations before reaching location $\mathcal{L}_u$. Denoting with $w(u)$ such a walk, the notation $\alpha_{w(u)n}$ specifies the number of appearances of SBS n in this walk. Based on Eqn. (6.3) and Eqn. (6.4), the *useful* portion of file $\mathcal{F}_u$ that the user downloads by an SBS $n \in \mathcal{L}_u$ at slot $\mathcal{T}_u$ is equal to $y_{n\mathcal{F}_u}^{\alpha_{w(u)n}}$. Then, the important point for this relation to hold is to define the state weights as follows:

Weight function Ω. The weight of a state $u \in \mathcal{S}$ is given by:

$$\Omega(u) = \begin{cases} 0, & \text{if } u = 0 \\ \sum_{n \in \mathcal{N}_{\mathcal{L}_u}} y_{n\mathcal{F}_u}^{\alpha_{w(u)n}}, & \text{otherwise} \end{cases} \tag{6.18}$$

Hence, the total weight of a $d-$step walk on the chain, which we model with a random variable Y, specifies the total portion of the requested file that a user manages to download by the encountered SBSs within the delay deadline. If $Y \geq 1$, the amount of downloaded data suffices to recover the requested file; otherwise the request is redirected to the MBS. The objective function in (6.6) can be written as the probability that Y is lower than 1, i.e., $\Pr[Y < 1]$. In the following, we show how the presented framework can be used to optimize the caching policy.

6.4.2 Upper bound on the objective function

We start with the following large deviation inequality.

Lemma 6.2. *(Hoeffding's Inequality for Markov Chains [114]).*
Consider an ergodic Markov chain and a t-step random walk starting from an initial distribution ϕ with total weight Y. For any $\delta \in [0, 1]$, there exists some constant c such that:

$$\Pr[Y \leq (1-\delta)\mu t] \leq c||\phi||_\pi \exp\left\{-\frac{\delta^2 \mu t}{72T}\right\} \tag{6.19}$$

where π is the stationary distribution, μ is the expected weight of the walk with respect to π, T is the mixing time, and $||\phi||_\pi$ indicates the $\pi-$norm of the vector ϕ.

Clearly, the Markov chain that we constructed in the previous subsection is *ergodic*, i.e., it is possible to go from every state to every state within a finite number of steps. Hence, we can use Lemma 6.2 to upper bound the objective function in (6.6), i.e., $\Pr[Y < 1]$. Specifically, for $t = d$, $\delta = 1 - 1/(\mu d)$ and $\mu d \geq 1$, and using the inequality $\Pr[Y < 1] \leq \Pr[Y \leq 1]$, we can show that:

$$\Pr[Y < 1] \leq c||\Phi||_{\pi} \exp\left\{ - \frac{\mu d + \frac{1}{\mu d} - 2}{72T} \right\} \tag{6.20}$$

Besides of the constant c, the value of which is given in [114], the initial probability distribution Φ defined in the previous subsection, and the delay deadline d, the upper bound depends on: (i) the stationary distribution π, (ii) the expected weight of a walk μ and (iii) the mixing time T. In the rest of this subsection, we formally define these values.

(i) Stationary distribution π. The stationary distribution π is a probability distribution vector on the states that is unchanged by the operation of transition matrix M on it, i.e.,

$$\pi = \pi M \tag{6.21}$$

Due to the special structure of this chain, we can compute the stationary probability for a state $u \neq 0$ as follows:

$$\pi(u) = \pi(0) \cdot M_{0,S_1} \cdot M_{S_1,S_2} \cdot \cdots \cdot M_{S_l,u} \tag{6.22}$$

where $(S_1, S_2, \ldots, S_l)$ denotes the intermediate states on the walk starting from the root until state u. Besides, since π is a probability distribution, it holds that:

$$\sum_{u \in \mathcal{S}} \pi(u) = 1 \tag{6.23}$$

By combining (6.22) and (6.23), we can show that:

$$\pi(0) = \frac{1}{1 + \alpha d} \tag{6.24}$$

(ii) Expected weight of a $d-$step walk μ. Let us first denote with $\Pr[y_{nf}^k]$ the probability with respect to π of reaching *any* state u with $n \in \mathcal{L}_u$, $f = \mathcal{F}_u$ and $\alpha_{w(u)n} = k$. For example, in Figure 6.3 there are three states with weight y_{11}^1 (the states corresponding

to marked rectangles) and hence it should be:

$$\begin{aligned}\Pr[y_{11}^1] = &\pi(0)\alpha p_1 \lambda_{11} \\ &+ \pi(0)\alpha p_2 \lambda_{21} q_{21} \\ &+ \pi(0)\alpha p_2 \lambda_{21} q_{22} q_{21}\end{aligned} \tag{6.25}$$

Then, by definition, the expected weight of a $d-$step random walk is equal to:

$$\mu(y) = \sum_{n=1}^{N} \sum_{f=1}^{F} \sum_{k=1}^{d} \Pr[y_{nf}^k] y_{nf}^k \tag{6.26}$$

where we have explicitly expressed μ as a function of y variables to capture this dependency.

(iii) Mixing time T**.** Following [114], we define $T = \min\{t : \max_q ||qM^t - \pi||_{TV} \leq \frac{1}{8}\}$. Here, q is an arbitrary initial distribution over $\mathcal{S}$. For two distributions q and q', it is: $||q - q'||_{TV} = \max_{A \subseteq \mathcal{S}} |\sum_{i \in A} q_i - \sum_{i \in A} q_i'|$.

6.4.3 Distributed algorithm

Since it is NP-Hard to directly minimize the objective function in (6.6), we take an alternate approach and minimize its upper bound in (6.20). From this approach, we obtain an approximate solution, together with a guaranteed upper bound on the achieved probability of requests routed to MBS.

Lemma 6.3 shows that minimizing this bound is equivalent to maximizing the expected weight $\mu(y)$.

Lemma 6.3. *Let* $g(y) = c||\phi||_\pi \exp\left\{-\frac{\mu(y)d + \frac{1}{\mu(y)d} - 2}{72T}\right\}$. *Then,* $_{(x,y)\in\mathcal{A}}\{g(y)\} = \text{argmax}_{(x,y)\in\mathcal{A}}\{\mu(y)\}$, *where* $\mathcal{A} = \{x, y : x_{nf} \in [0,1], y_{nf}^k \in [0,1]\ \forall n \in \mathcal{N},\ f \in \mathcal{F},\ k \in \{1, 2, \ldots, d\}$ *and constraints (6.7)-(6.8) and (6.12)-(6.13) are satisfied*$\}$.

Proof. Let $(x^*, y^*) =_{(x,y)\in\mathcal{A}} \{g(y)\}$, i.e., $g(y^*) \leq g(y)\ \forall (x, y) \in \mathcal{A}$. Dividing by $c||\phi||_\pi$ and then taking the logarithm on both sides preserves the inequality as $c > 0$, $||\phi||_\pi > 0$ and log(.) is increasing function. Hence, we obtain:

$$-\frac{\mu(y^*)d + \frac{1}{\mu(y^*)d} - 2}{72T} \leq -\frac{\mu(y)d + \frac{1}{\mu(y)d} - 2}{72T} \tag{6.27}$$

Dividing by $-72T \leq 0$ and then adding 2 on both sides yields:

$$\mu(y^*)d + \frac{1}{\mu(y^*)d} \geq \mu(y)d + \frac{1}{\mu(y)d} \tag{6.28}$$

However, it holds that: $\mu(y)d \geq 1$, resulting that: $\mu(y^*)d \geq \mu(y)d \ \forall (x,y) \in \mathcal{A}$, which completes the proof. □

Based on Lemma 6.3, the caching problem becomes:

$$\max_{x,y} \ \mu(y) \tag{6.29}$$
$$s.t. \ (6.7), (6.8), (6.12), (6.13)$$

The above problem is more tractable than the original caching problem. By the structure of $\mu(y)$, that is described in Eqn. (6.26), and the above constraints we can show that the caching decisions at an SBS do not affect the rest. Hence, we can *decompose* this problem to N independent subproblems, one for each SBS, and solve them in a *distributed* manner. The problem for an SBS $n \in \mathcal{N}$, which we refer to as $\mathcal{P}_n$, can be expressed as follows:

$$\mathcal{P}_n: \ \max_{x_n,y_n} \ \sum_{f=1}^{F}\sum_{k=1}^{d} \Pr[y_{nf}^k] y_{nf}^k \tag{6.30}$$
$$s.t. \ \sum_{f\in\mathcal{F}} s_f x_{nf} \leq C_n \tag{6.31}$$
$$x_{nf} \in [0,1], \forall f \in \mathcal{F} \tag{6.32}$$
$$y_{nf}^k \in [0, \frac{B_n}{s_f}], \ \forall f \in \mathcal{F}, k = 1, ..., d \tag{6.33}$$
$$\sum_{k=1}^{d} y_{nf}^k \leq x_{nf}, \ \forall f \in \mathcal{F} \tag{6.34}$$

where x_n and y_n denote the variables in x and y for SBS n. The objective function and the constraints of this problem are linear with respect to the optimization variables. Hence, it can be efficiently solved using standard *linear optimization techniques* [56]. Going one step further, we show that $\mathcal{P}_n$ falls into a class of knapsack problems with *known solution structure*, alleviating the need for applying linear optimization techniques. Specifically, we will prove the following lemma.

Lemma 6.4. *The optimal solution of problem $\mathcal{P}_n$ can be computed in $O(F \cdot d \cdot \log(F \cdot d))$ time.*

Proof. *Fractional knapsack problem* asks for placing fractions of items of different values and weights in a knapsack of limited capacity in a way that maximizes the aggregate value of items placed in it [115]. The problem $\mathcal{P}_n$ can be translated to a *restricted* version

Algorithm 6.1: Mobility-aware caching algorithm

Input : An instance of the $\mathcal{P}_n$ problem.
Output: The optimal solution x_n, y_n.

1 $value^k_{nf} \leftarrow \Pr[y^k_{nf}],\ \forall f \in \mathcal{F}, k \in \{1, \ldots, d\}$;
2 $weight^k_{nf} \leftarrow s_f,\ \forall f \in \mathcal{F}, k \in \{1, \ldots, d\}$;
3 $y^k_{nf} \leftarrow 0,\ \forall f \in \mathcal{F}, k \in \{1, \ldots, d\}$;
4 $\mathcal{D} \leftarrow \mathcal{F} \times \{1, \ldots, d\}$;
5 **for** $i = 1, 2, \ldots, F \cdot d$ **do**
6 $\quad (f^*, k^*) \leftarrow \text{argmax}_{(f,k)\in\mathcal{D}} \frac{value^k_{nf}}{weight^k_{nf}}$;
7 $\quad y^{k^*}_{nf^*} \leftarrow \min\left\{\frac{B_n}{s_{f^*}},\ 1 - \sum_{k=1}^{d} y^k_{nf^*},\ C_n - \sum_{f\in\mathcal{F}}\sum_{k=1}^{d} y^k_{nf}\right\}$;
8 $\quad \mathcal{D} \leftarrow \mathcal{D} \setminus (f^*, k^*)$;
9 $\quad$ **if** $\sum_{f\in\mathcal{F}}\sum_{k=1}^{d} y^k_{nf} = C_n$ **then**
10 $\quad\quad$ break;
11 $x_{nf} = \sum_{k=1}^{d} y^k_{nf},\ \forall f \in \mathcal{F}$;

of the fractional knapsack problem in which there exist $F \cdot d$ items, one item for each variable $y^k_{nf},\ f \in \mathcal{F},\ k = 1, 2, \ldots, d$ and a knapsack of capacity C_n. The value of the item corresponding to y^k_{nf} is $\Pr[y^k_{nf}]$ and it's weight is s_f. The item placement must also satisfy the following two restrictions: **Restriction 1**: At most B_n/s_f fraction of the item corresponding to variable y^k_{nf} can be placed in the knapsack, $\forall f, k$. **Restriction 2**: The total amount of items corresponding to variables $y^k_{nf},\ k = 1, 2, \ldots, d$ placed in the knapsack must be less or equal to 1, $\forall f$.

The optimal solution of this knapsack-type problem can be attained by a simple scheme that iteratively places a fraction of the item with the highest ratio of value/weight in the knapsack until the knapsack becomes full [115]. At each iteration, the scheme ensures that the item placement satisfies the above two restrictions. The knapsack solution can be mapped to a solution to the problem $\mathcal{P}_n$ such that every y^k_{nf} variable takes as value the fraction of the associated item placed in the knapsack and x_{nf} takes as value the sum: $\sum_{k=1}^{d} y^k_{nf}$. This procedure is summarized in Algorithm 6.1.

Here, $value^k_{nf}$ and $weight^k_{nf}$ denote the value and weight of the item corresponding to variable y^k_{nf} respectively (lines 1-2). Operator $\times$ denotes the cartesian product of two sets. Collection $\mathcal{D}$ includes all the (f, k) pairs for which item y^k_{nf} has not been picked yet (line 4). At every iteration, Algorithm 6.1 picks the pair $(f^*, k^*) \in \mathcal{D}$ with the largest ratio value/weight (line 6). Then, a fraction of item $y^{k^*}_{nf^*}$ will be placed in the knapsack satisfying the two restrictions (line 7). The algorithm terminates when all the items are picked or the knapsack becomes full (line 9).

In the worst case, all the $F \cdot d$ items will be picked. Since the items are picked in decreasing order of their ratios of value/weight, sorting all these ratios is required. Hence, the complexity of Algorithm 6.1 will be $O(F \cdot d \cdot \log(F \cdot d))$ [59], which completes the proof of lemma 6.4. □

6.4.4 Implementation considerations

We need to emphasize that the overhead of cooperation among SBSs and MBS for serving the user requests may affect the efficiency of Algorithm 6.1. For example, the operator may use techniques like network initiated offloading [116] to dynamically decide which SBS will serve each request taking into account the cached content. This may cause additional latency for signaling among SBSs and MBS [117] which can be to the detriment of the mobile users. An operator can estimate the overhead latency, e.g., by processing previous time period statistics. This information can be used to estimate the amount of data delivered in a slot to a user. The latter is captured in our model by the parameter B_n for each SBS n that is passed as input to Algorithm 6.1.

A second aspect that we need to consider is that several Video-on-Demand sites (YouTube, Netflix, Hulu, etc) make use of DASH-based [118] (or similar) adaptive streaming approaches in the late years. Optimizing the quality of streaming experience creates a far more difficult challenge for caching in SBSs since it involves new metrics such as start-up delay, video stalls, frame rate and spatial resolution. This is a different approach from our main objective, since the method we describe centers on offloading the macro-cell networks. The latter is particularly important during periods of peak traffic when scalability issues are raised. Even if an adaptive video streaming protocol is applied, we stress that our method requires only a small delay for streaming to start. This is captured by the delay deadline d. As we show in the next section, our approach achieves significant gains, in terms of reduction in macrocell's load, delaying video viewing by less than 1 minute. The latter is an often acceptable video start-up delay.

6.5 Performance evaluation

In this section, we evaluate the performance of the proposed algorithm using real traces of human mobility patterns and requests for Youtube videos. Overall, we find that moving from a conventional caching algorithm to one enhanced with mobility-awareness reduces the load of the macro-cell network, and this heavily depends on cache sizes, delay deadline and density of SBS deployment. In the best scenario, an operator can improve its bottom line by 65% delaying data transfer by 1 minute. In the rest of this

section, we discuss these results in detail; we begin by describing the algorithms used in the later evaluations.

6.5.1 Algorithms

Throughout the evaluation, we compare the performance of three algorithms:

1. *Max-popularity*: Each SBS stores in its cache the locally most popular files independently from the others.
2. *Femtocaching [33]*: All users are assumed to be stationary during the evaluation and caching decisions are made based on the initial distribution of the users in the cell. Particularly, the algorithm starts with all the caches being empty. Iteratively, it performs the placement of a file to a cache that minimizes the probability of requests served by MBS, i.e., $\sum_{l\in\mathcal{L}} p_l \sum_{f\in\mathcal{F}} \lambda_{lf} 1_{\{\sum_{n\in\mathcal{N}_l} x_{nf}<1\}}$. The procedure terminates when all the caches become full.
3. *Mobility-aware*: We apply Algorithm 6.1 to determine the caching policy for each SBS in a distributed manner.

The first two algorithms take caching decisions considering only user demand (i.e., λ_{lf} and p_l). The proposed algorithm considers also information about the users' motility patterns (i.e., $q_{ll'}$ values) and places MDS-encoded file segments at the SBSs instead of entire file copies.

6.5.2 Mobility model

We evaluate the performance of the described schemes using the measured trace of mobility patterns released by the Wireless Topology Discovery project [119]. This trace contains information from approximately 275 PDA users for an 11 week period between September 22, 2002 and December 8, 2002. Specifically, each active user records every 20 seconds all the WiFi access points (APs) that are detected by its device. Due to the short distance between APs, a user may sense more than one APs at the same time. In total, more than 400 APs are detected. For each one of these APs, its geographical location, described by a pair of (X,Y) values (measured in meters), is also recorded.

We focus on a certain subarea in [119] with dense AP deployment depicted in Figure 6.4. To facilitate presentation, we shifted the point (X,Y)=(1698270,259950) to the zero coordinates. This area includes $N = 15$ APs in total. In our evaluation, we substitute SBSs for WiFi APs as in [49]. This is a reasonable approximation for pico-cells, since

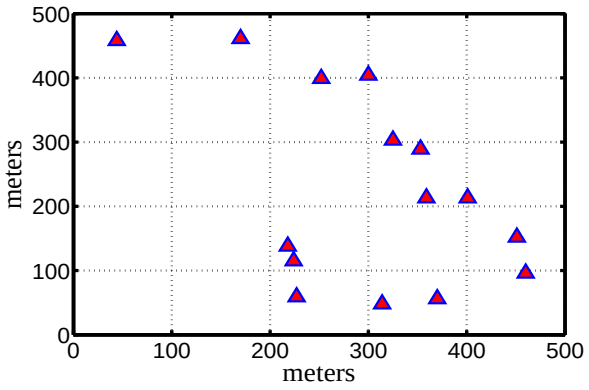


FIGURE 6.4: A 500m× 500m area with 15 APs (triangles) in [119].

the radius of the latter usually resembles that of the WiFi APs ($\sim$ 100 meters [106]). Importantly, due to the fact that operators typically keep the datasets of mobility across their base stations confidential, it would be difficult to acquire such information. In contrast, the trace we use is publicly available online. Hence, any caching algorithms that will be developed in the future can be directly compared with the proposed one under the same network settings. This is of high importance since it ensures that our work is not detached from future research efforts.

We focus on the busiest day, namely the day of 16 October 2002, and keep the peak time statistics, i.e., between 18pm and 23pm. For each subset of SBSs that concurrently cover a user we create a distinct location, which results into $L = 185$ locations in total. We also set the time slot duration to be 20 seconds. Then, for each location l, we set the p_l probability to be the portion of slots that users visit location l. In order to compute $q_{ll'}$, we divide the number of sequential visits to locations l and l' over the total number of visits to l. If a user becomes inactive by the end of a slot, we assume that he remains in the same location, and hence we increase the value of q_{ll}, where l is the location recorded last.

Intuitively, the benefits of applying a mobility-aware caching scheme instead of a conventional one are higher when users move rapidly in and out of the SBS coverage areas. Therefore, it is crucial to answer how often this occurs. Figure 6.5 aims to shed light on this question by showing the cumulative distribution function (CDF) of the number of SBSs detected by a user within a deadline of $d \in \{1, 2, 3, 4, 5\}$ slots. Here, for $d = 1$, all the requests are satisfied within a single slot, and hence all the users can download data only by the SBSs detected in this slot. However, for $d > 1$, the users can detect additional SBSs encountered in subsequent slots until the deadline expires. The average number of detected SBSs increases from 2.76 for $d = 1$ up to 6.87 for $d = 5$ slots. This is a drastic increase, bearing in mind that the extra time interval is only a few tens of

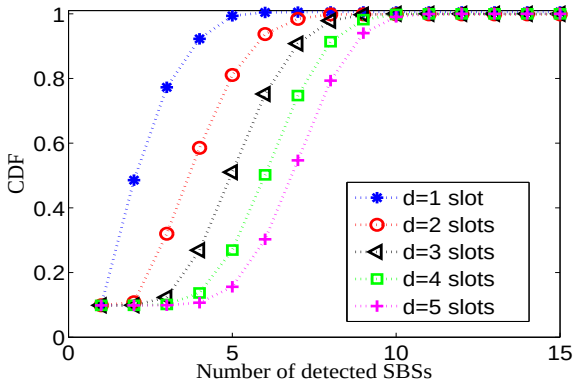


FIGURE 6.5: Cumulative distribution function of the number of SBSs detected during a user walk in [119].

seconds long. Clearly, all the SBSs that are detected during this extra time interval can be used to deliver content to the user if they have cached parts of the requested file. Hence, *even for a short delay deadline d, e.g., a few minutes, a mobility-aware caching scheme can potentially offload more traffic from the macro-cell network compared to a conventional scheme.*

6.5.3 Demand model

In order to model user demand, we use a measured trace of YouTube requests from a study performed at Amherst campus, University of Massachusetts, in 2008 [120]. The trace records for each request arising from the (wired) campus network, its exact time and a unique identifier of the requested video. We use the trace data for the two consecutive weeks starting from January 29th, 2008. The number of requests for the $10,000$ most popular videos is presented in Figure 6.6. In our evaluation, we consider the same video library ($F = 10,000$) and set the λ_{lf} probabilities for each location l based on the popularity of video f in the trace as in [33].

6.5.4 Evaluation results

Throughout the evaluation, we set all video files to be of size 40 MB, which is reasonable assuming a screen size 640×360, flash encoding and a few minutes ($3-4$ min) playback time. Unless otherwise specified, each SBS n is endowed with a cache of size $C_n = C\ \forall n \in \mathcal{N}$ that can store up to 10% of the entire video library size. To set the B_n values, we follow the real bandwidth measurements in [121], which report an average bit-rate between a user and an SBS equal to 8 mbps. Hence, we upper bound the per-slot amount of data that an SBS n can deliver to a user by $B_n = 20MB$. The performance

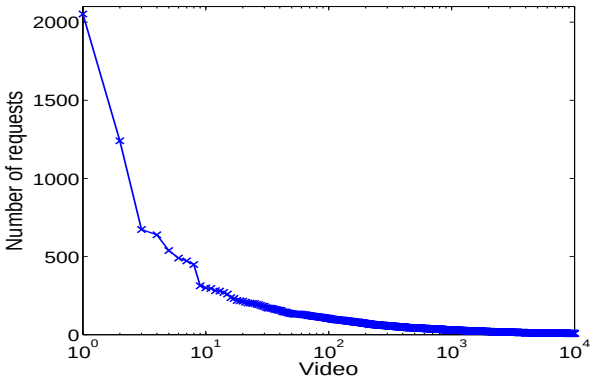


FIGURE 6.6: Number of requests for videos in Youtube trace [120].

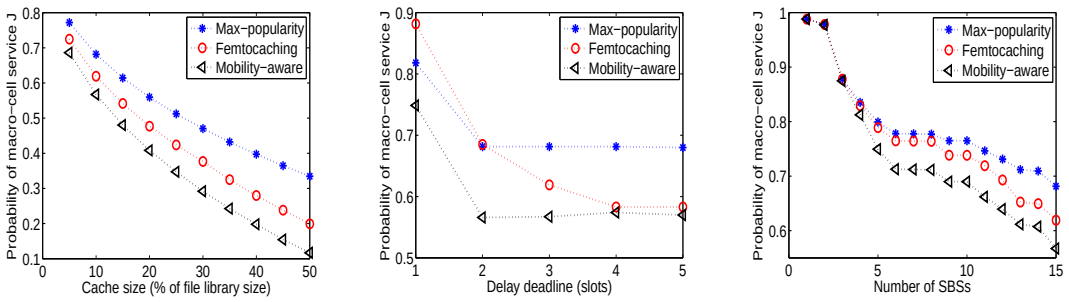
criterion we use is the probability that a request is served by the macro-cell network denoted with J (Eqn. (6.5)).

Before we proceed with the evaluation results, let us remark that for the algorithms' implementation we used the C++ language in the Visual Studio environment.

Impact of cache sizes: We first compare the probability that a request is served by the macro-cell network (J) achieved by the presented algorithms for different cache sizes. In the experiment in Figure 6.7(a), cache sizes span a wide range of values, starting from 5% to 50% of the entire file library size, reflecting different operator conditions. As expected, increasing the cache sizes reduces J for all algorithms, since more files become available for download from the SBSs. Femtocaching consistently outperforms Max-popularity algorithm. This can be explained from the fact that the latter simply stores the C most popular files at all the SBSs. However, in dense SBS deployments, users often access multiple SBSs and each sees a distributed cache that is the union of the respective caches. Clearly, such users would benefit if some of the SBSs stored different (less popular) files, since they could find more files at the accessed SBSs. Femtocaching algorithm identifies such cases by considering the overlapping SBS coverage areas, captured by the $\mathcal{N}_l$ values.

Besides of the superiority of Femtocaching over Max-popularity, Figure 6.7(a) comments also on the inefficiency of the existing caching policies that are designed for static networks. Namely, in realistic environments where users move rapidly from SBS to SBS, mobility impacts the performance of the caching policy. Mobility-aware is the only algorithm among the three that exploits user mobility, captured by the $q_{ll'}$ values. In our experiment, *Mobility-aware performs markedly better than Max-popularity and Femtocaching, where the gains increase with cache sizes reaching* 65% *and* 41% *respectively.*

Impact of delay deadline: Figure 6.7(b) shows how the performance of the presented algorithms depends on the delay deadline d. In this experiment, the deadline varies



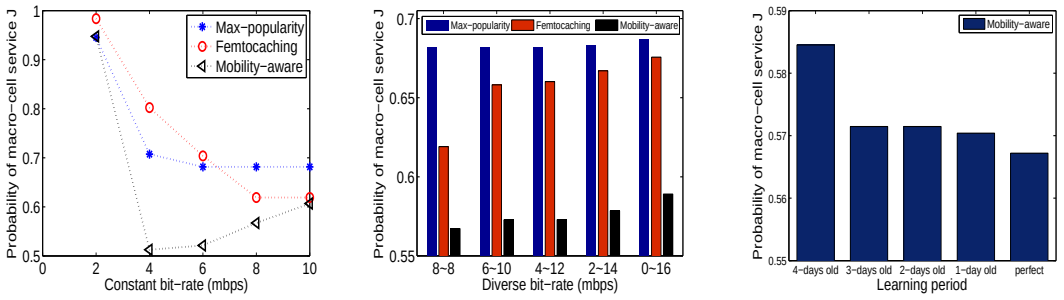
(a) (b) (c)

FIGURE 6.7: Probability that a request is served by the macro-cell network for different values of: (a) the cache size per SBS C, (b) the delay deadline d and (c) the number of SBSs N.

within $\{1, 2, 3, 4, 5\}$ slots, i.e., $\{20, 40, 60, 80, 100\}$ seconds. This is an often acceptable video start-up delay. We observe that as d increases, probability J decreases for all the algorithms, since users have more contact opportunities with the SBSs. The performance of Max-popularity saturates at $d = 2$, since from this point and above users download all the most popular files that are fully replicated at the SBSs, but none of the rest files. On other hand, Femtocaching stores different files across the SBSs, and hence users can download more files as d increases. Among the three algorithms, Mobility-aware exploits better the contact opportunities, since for a sequence of SBSs that are frequently visited one after the other, coded parts of a file are spread to all the SBSs instead of storing complete file copies at some of them. This increases the number of potential sources from which a user can obtain data and can potentially decrease probability J. Specifically, we find that *Mobility-aware outperforms Max-popularity and Femtocaching for all values of d, where the gains can be up to* 16% *and* 15% *respectively.*

Impact of SBS density: We explore how the density of SBSs impacts the results in Figure 6.7(c). Specifically, we consider the topology depicted in Figure 6.4, but keep a randomly chosen subset of the SBSs. We observe that as the number of SBSs increases, the probability J decreases for all the algorithms, since users encounter more SBSs within the deadline. Mobility-aware consistently outperforms the rest algorithms, especially for high number of SBSs. For example, the gains over Max-popularity and Femtocaching are very close to zero when there exist five SBSs, but increase to 16% and 8% respectively when all the fifteen SBSs are considered. As a takeaway, *the superiority of Mobility-aware over the alternate algorithms that were examined, is more pronounced for dense SBS deployments.*

Impact of SBS bit-rate: We analyze the impact of the available bit-rate between SBSs and users on the algorithms' performance in Figure 6.8(a). Specifically, we vary



(a) (b) (c)

FIGURE 6.8: Probability that a request is served by the macro-cell network for (a) constant and (b) diverse bit-rate between SBSs and users. (c) The impact of learning period.

the bit-rate from 2 to 10 mbps. As expected, increasing the bit-rate decreases the probability J achieved by Max-popularity and Femtocaching, since the SBSs can transfer more data during the contacts with users. For low bit-rates, i.e., when the system is in overloaded conditions, storing the most popular files at all SBSs offloads more the macro-cell than the Femtocaching algorithm does. In contrast, as the bit-rate increases, the need for diversing the cached content becomes more apparent and, hence, Femtocaching outperforms Max-popularity. Interestingly, *Mobility-aware outperforms the Max-popularity and Femtocaching algorithms for all bit-rates, a gap being up to* 27% *and* 36% *respectively.*

The numerical results presented so far assume constant bit-rate between SBSs and users. Nevertheless, the bit-rate often varies over time, e.g., due to variations in channel quality, temporal interference, congestion effects, etc. To capture the above dynamics of the wireless medium, we synthesize a variety of scenarios that differ in the way that the bit-rate is set. Specifically, the bit-rate value is randomly drawn from a range. We denote with $a \sim b$ the scenario when the bit-rate is randomly generated within $[a, b]$. Figure 6.8(b) compares the performance of the three presented algorithms for the scenarios that $a \sim b$ is $8 \sim 8$, $6 \sim 10$, $4 \sim 12$, $2 \sim 14$ and $0 \sim 16$ (mbps). We observe that the probability J increases more-or-less for all the algorithms as the bit-rate becomes more diverse. This can be explained by the fact that, for diverse bit-rates, some users are served with very low bit-rate by the SBSs, while others are served with higher bit-rate than they need. Interestingly enough, we notice that *the performance of Femtocaching is rather sensitive to the diversity of bit-rates, while that of the Max-popularity and Mobility-aware is quite stable.* Mobility-aware is always better than the rest algorithms, with the gains being up to 14.2% and 12.8% when compared to the Max-popularity and Femtocaching algorithm respectively.

Impact of learning period: The numerical results presented so far assume perfect

knowledge of user mobility behavior, i.e., the exact probability values p_l and $q_{ll'}$ $\forall l, l' \in \mathcal{L}$ are known. In practice though, these values are predicted by analyzing statistics of a previous time period (learning period). Clearly, the accuracy of such predictions impacts the efficiency of the Mobility-aware caching algorithm. Figure 6.8(c) aims to shed light on this issue by evaluating the performance of Mobility-aware on October 16^{th}, 2002 for different learning periods. Here, statistics are taken from each one of the previous four days; these are referred to as 1-, 2-, 3- and 4-days old. Interestingly, we see that the performance of Mobility-aware is rather stable within these days (less than 3% loss compared to the case of perfect knowledge), which indicates that the user mobility behavior changes slowly in time. This is very important as it shows that, *for an operator periodically tracking user movements in the small-cell network, substantial performance benefits can be realized from applying our algorithm.*

Main takeaways: User mobility impacts the efficiency of caching policies in hyper-dense SBS deployments. The proposed mobility-aware caching algorithms leverage predications about the user mobility patterns to effectively cache file segments to multiple SBSs. Trace-driven numerical results show that the proposed approach can offload up to 65% more traffic of the macro-cells than existing caching algorithms.

Chapter 7

Conclusions and Future work

Contents

7.1 Conclusions

In this thesis, we studied the problem of *caching* content in wired and wireless networks aimed at optimizing network operator costs and user performance. Our work combines strong theoretical contributions with a careful empirical data analysis from real operators and users.

Specifically, in Chapter 2, we proved the NP-Hardness of the caching problem in multiple-level hierarchical networks, uncovered a tractable special case of caches installed on a single hierarchy path, and developed an algorithm achieving a provably better approximation ratio than the best-known counterparts. Numerical results for typical popularity distributions demonstrated significant performance improvements over conventional caching schemes, which are more pronounced when the content popularity distribution is steep and the cache capacities at the upper hierarchy levels are large.

In Chapters 3-6, we moved to emerging architectures that enable caching at the wireless edge (macro-cell, pico-cell, femto-cell base stations and WiFi APs). We first showed how the presence of massive content demand perplexes the problem and developed approximation algorithms that make caching decisions jointly with the routing of content among

caches (Chapter 3). Then, we extended our framework for supporting diverse requests for video contents, where each video can be delivered in different qualities (Chapter 4). Finally, we proposed caching algorithms that exploit the broadcast nature of wireless medium (Chapter 5) and the predictability of user mobility (Chapter 6). Simulation results indicated that for an operator having at his disposal a few days-old statistics of content requests and user mobility, thus being able to infer future request and mobility patterns, our approach can serve as an important tool for removing bottlenecks on the wired backbone networks and extracting maximum benefit in network performance.

7.2 Future work

Our goals for future work include further research on cache management and routing methods, as well as evaluating them on new, larger datasets of content request patterns. Specifically, we intend exploring the following open issues:

7.2.1 Incomplete information

First, we emphasize that the caching schemes presented in this thesis assume the presence of a bounded-size library of content files (e.g., movies, TV shows, news) that is refreshed relatively slowly (e.g., on a daily or weekly basis) and the respective user demand is perfectly known. Hence, the effectiveness of our caching schemes depends on our ability to understand and predict demand across users. A method to achieve this is by analyzing previous-time user preferences/ratings for content to infer on future events. This is even more challenging when considering video content, since often some newly generated videos become viral within a very short time, and hence the history of preferences/ratings is limited. The classical techniques for making such inferences are based on hierarchical Bayesian models and require high computational costs which render them infeasible for large-scale applications. This process can be fastened by leveraging tools from machine learning, such as collaborative filtering techniques [53], [54]. Alternatively, the average demand can be learned online based on the instantaneous demand on a base station, in which case the caching problem can be modeled as a multi-armed bandit problem [55]. Clearly, additional methods are needed to further reduce the computational time of demand forecasting.

7.2.2 Communication overhead

There is also an overhead of cooperation and coordination among cache-nodes that needs to be taken into account when implementing caching mechanisms. In fact, if multiple nodes collaborate to cache some content segments, this requires extra metadata management and may increase user experienced delay. This is more crucial in highly-mobile networks where users are rapidly handed-off between the base stations. This issue is outside the scope of our study, and we leave it as a future work.

7.2.3 Conflicting objectives

Finally, we emphasize that operators and users have conflicting objectives; operators aim at reducing their servicing costs, whereas user satisfaction degrades with increasing the content delivery delay. Due to the close proximity between users and caches, the delay is typically lower for users served by the caches than the rest. Following this observation, in Chapters 2-3 we maximized the portion of requests that are served by the caches (*cache hit rate*). In Chapter 4, we explicitly considered a balanced objective function of user delay and operator cost. In Chapter 5, we captured the minimum acceptable level of user satisfaction by the time period of the multicast service within which all pending user requests were collected and served by the multicast stream. Finally, in Chapter 6, we associated all user requests with a time deadline. When this deadline expired, all requests left unserved by the encountered base stations were redirected to the macro-cell network. Despite our analysis, a more detailed study of the tradeoff between the aggregate transmission cost of a packet and the delay of its delivery is still necessary.

Appendix

[**A. Proof of Lemma 2.1**]

Lemma 2.1. In the optimal caching policy, the files stored on $\mathcal{P}_n$ path are disjoint, i.e., no two nodes on $\mathcal{P}_n$ store the same file, $\forall n \in \mathcal{L}$.

Proof. Let us consider an optimal caching policy x^o with two nodes on $\mathcal{P}_n$ storing the same file, say file f. Since the two nodes belong to the same path, they are located in different levels in the hierarchy, with one of them being higher than the other. We denote with n_h the higher-level node and with n_l the lower-level node. Due to the hierarchical network structure, all the requests for file f that traverse n_l node pass through n_h node before reaching servers. Hence, removing file f from the cache of n_l node has no impact on the server load as long as file f is stored at n_h node. Leveraging the cache space freed after removal to add another file f', where f' was not previously stored in any of the caches in $\mathcal{P}_n$, would decrease server load at least by $\lambda_{nf'} > 0$. This contradicts the assumption of optimality of x^o and completes the proof. □

[**B. Proof of Theorem 2.12**]

Theorem 2.12. In a two-level hierarchy with two leaves, Algorithm 2.1 finds the optimal solution to the caching problem in $2F \log F + C_0F(C_1 + C_2 + 2C_0)$ time.

At the first iteration, Algorithm 2.1 places in the root cache the file that results the highest value of the objective function in (2.16). Clearly, if the root cache is of size $C_0 = 1$, Algorithm 2.1 will output the optimal solution. For the general case of $C_0 > 1$, Algorithm 2.1 will place in the root additional files in a similar manner, restricting at each iteration that the files placed in the root at previous iterations will be also part of the current solution. Therefore, in order for Algorithm 2.1 to be optimal, it suffices to show that this restriction does not result in a degradation of overall performance. This is proved in the following lemma.

Lemma .1. *Consider two instances of the caching problem in a two-level hierarchy with* $L = 2$ *leaves, differing only in the size of the root cache;* k *for the first instance and* $k+1$

for the second instance. Then, the set of files stored in the root cache by the optimal caching policy for the first instance is a subset of the files stored for the second instance.

Proof. Without loss of generality, we set $k = 1$. Let α be the file stored in the root cache according to the optimal caching policy for the first problem instance. Also, let $M_n(s)$ be the C_n most popular files with respect to λ_n in $\mathcal{F}$ excluding the files in subset s, for $n = 1, 2$. Then, based on Lemma 2.1, leaf 1 stores the files in $M_1(\alpha)$ and leaf 2 the files in $M_2(\alpha)$. Hence, the total number of requests served by the caches is given by:

$$hit_1 = \lambda_{1\alpha} + \lambda_{2\alpha} + \sum_{f \in M_1(\alpha)} \lambda_{1f} + \sum_{f \in M_2(\alpha)} \lambda_{2f} \tag{1}$$

Let us consider two files β and γ, such as β is stored in the leaf cache 1 and γ in the leaf cache 2. In other words, $\beta \in M_1(\alpha) \setminus M_2(\alpha)$ and $\gamma \in M_2(\alpha) \setminus M_1(\alpha)$. Consider also a third file δ that is not stored in any of the leaf caches, i.e., $\delta \notin M_1(\alpha)$ and $\delta \notin M_2(\alpha)$. Due to the optimality of the caching policy the following three inequalities hold:

$$hit_1 \geq \lambda_{1\beta} + \lambda_{2\beta} + \sum_{f \in M_1(\beta)} \lambda_{1f} + \sum_{f \in M_2(\beta)} \lambda_{2f} \tag{2}$$

$$hit_1 \geq \lambda_{1\gamma} + \lambda_{2\gamma} + \sum_{f \in M_1(\gamma)} \lambda_{1f} + \sum_{f \in M_2(\gamma)} \lambda_{2f} \tag{3}$$

$$hit_1 \geq \lambda_{1\delta} + \lambda_{2\delta} + \sum_{f \in M_1(\delta)} \lambda_{1f} + \sum_{f \in M_2(\delta)} \lambda_{2f} \tag{4}$$

We now consider the second problem instance with root cache size 2. We assume that the Lemma does not hold, i.e., there exists a policy storing two different to α files in the root cache that offloads more requests than a policy storing α and any other file in it. We distinguish the following two cases depending on whether files β and γ are stored in the root cache:

Case 1: at most one of the files β and γ is stored in the root cache. Without loss of generality, let β be that file, and denote with δ the second file stored in the root cache. Then, the total number of requests served by the caches is given by:

$$hit_2 = \lambda_{1\beta} + \lambda_{2\beta} + \lambda_{1\delta} + \lambda_{2\delta} + \sum_{f \in M_1(\beta,\delta)} \lambda_{1f} + \sum_{f \in M_2(\beta,\delta)} \lambda_{2f} \tag{5}$$

Since δ is not included in any of the sets $M_1(\alpha)$ and $M_2(\alpha)$, we infer that:

$$M_1(\alpha) = M_1(\alpha, \delta) \tag{6}$$

$$M_2(\alpha) = M_2(\alpha, \delta) \tag{7}$$

Then, we use equality (5) to obtain:

$$\begin{aligned} hit_2 &\leq \lambda_{1\beta} + \lambda_{2\beta} + \lambda_{1\delta} + \lambda_{2\delta} + \sum_{f \in M_1(\beta)} \lambda_{1f} + \sum_{f \in M_2(\beta)} \lambda_{2f} \\ &\leq \lambda_{1\alpha} + \lambda_{2\alpha} + \lambda_{1\delta} + \lambda_{2\delta} + \sum_{f \in M_1(\alpha)} \lambda_{1f} + \sum_{f \in M_2(\alpha)} \lambda_{2f} \\ &= \lambda_{1\alpha} + \lambda_{2\alpha} + \lambda_{1\delta} + \lambda_{2\delta} + \sum_{f \in M_1(\alpha,\delta)} \lambda_{1f} + \sum_{f \in M_2(\alpha,\delta)} \lambda_{2f} \end{aligned} \tag{8}$$

where the first inequality is due to the definition of $M_1(.)$ and $M_2(.)$ sets. The second inequality is due to inequalities (1) and (2). The last equality is due to equalities (6)-(7). Based on inequality (8), the policy storing files α and δ in the root cache outperforms the policy storing files β and δ, which contradicts the assumption made.

Case 2: both files β and γ are stored in the root cache. Then, the total number of requests served by the caches is given by:

$$hit_2 = \lambda_{1\beta} + \lambda_{2\beta} + \lambda_{1\gamma} + \lambda_{2\gamma} + \sum_{f \in M_1(\beta,\gamma)} \lambda_{1f} + \sum_{f \in M_2(\beta,\gamma)} \lambda_{2f} \tag{9}$$

If file α is stored at both the leaf caches, it is equivalent (in terms of performance) of having it stored in the root cache. To see this, notice that swapping file α from the leaf caches with a file stored in the root cache, will result the same server load. To avoid such trivial cases, we distinguish the following two subcases:

Subcase 1: File α is not stored in leaf 1, i.e., $\alpha \notin M_1(\beta, \gamma)$.
By definition of $M_1(.)$, it should be also: $\alpha \notin M_1(\gamma)$. Given that we chose γ so as $\gamma \notin M_1(\alpha)$, we obtain:

$$M_1(\alpha) = M_1(\gamma) \tag{10}$$

Using (1) and (10), inequality (3) becomes:

$$\lambda_{1\gamma} + \lambda_{2\gamma} \leq \lambda_{1\alpha} + \lambda_{2\alpha} + \sum_{f \in M_2(\alpha)} \lambda_{2f} - \sum_{f \in M_2(\gamma)} \lambda_{2f} \tag{11}$$

Then, we use equality (9) to obtain:

$$\begin{aligned} hit_2 &\leq \lambda_{1\beta} + \lambda_{2\beta} + \lambda_{1\gamma} + \lambda_{2\gamma} + \sum_{f \in M_1(\beta,\gamma)} \lambda_{1f} + \sum_{f \in M_2(\gamma)} \lambda_{2f} \\ &\leq \lambda_{1\beta} + \lambda_{2\beta} + \lambda_{1\alpha} + \lambda_{2\alpha} + \sum_{f \in M_1(\beta,\gamma)} \lambda_{1f} + \sum_{f \in M_2(\alpha)} \lambda_{2f} \\ &= \lambda_{1\beta} + \lambda_{2\beta} + \lambda_{1\alpha} + \lambda_{2\alpha} + \sum_{f \in M_1(\alpha,\beta,\gamma)} \lambda_{1f} + \sum_{f \in M_2(\alpha)} \lambda_{2f} \\ &= \lambda_{1\beta} + \lambda_{2\beta} + \lambda_{1\alpha} + \lambda_{2\alpha} + \sum_{f \in M_1(\alpha,\beta,\gamma)} \lambda_{1f} + \sum_{f \in M_2(\alpha,\beta)} \lambda_{2f} \end{aligned} \tag{12}$$

where the first inequality is due to the definition of $M_2(.)$. The second inequality is due to inequality (11). The first equality is because in this subcase it holds that $\alpha \notin M_1(\beta, \gamma)$, which results to $M_1(\beta, \gamma) = M_1(\alpha, \beta, \gamma)$. The second equality is because we chose γ so as $\beta \notin M_2(\alpha)$. Based on inequality (12), the policy storing files α and β in the root cache outperforms the policy storing files β and γ, which contradicts the assumption made.

Subcase 2: File α is not stored in leaf 2, i.e., $\alpha \notin M_2(\beta, \gamma)$.
Similarly to subcase 1, we obtain:

$$M_2(\alpha) = M_2(\beta) \tag{13}$$

and

$$\lambda_{1\beta} + \lambda_{2\beta} \leq \lambda_{1\alpha} + \lambda_{2\alpha} + \sum_{f \in M_1(\alpha)} \lambda_{1f} - \sum_{f \in M_1(\beta)} \lambda_{1f} \tag{14}$$

Then, we use equality (9) to obtain:

$$hit_2 \leq \lambda_{1\alpha} + \lambda_{2\alpha} + \lambda_{1\gamma} + \lambda_{2\gamma} + \sum_{f \in M_1(\alpha,\gamma)} \lambda_{1f} + \sum_{f \in M_2(\alpha,\beta,\gamma)} \lambda_{2f} \tag{15}$$

which shows that the policy storing files α and γ in the root cache outperforms the policy storing files β and γ, which contradicts the assumption made.

Hence, for every case, we showed that one of the two files stored in the root cache by the optimal caching policy is the same file stored in it when the size of the root cache is 1, which completes the proof. □

[C. Proof of Lemma 4.3]

Lemma 4.3. The problem P_n is polynomial-time reducible to the MCK problem.

Proof. Given an instance of the P_n problem, we construct the equivalent instance of the MCK problem as follows: There is a knapsack of size equal to S_n and the item classes $E_1, E_2, ..., E_V$, each with Q items. The i^{th} item in class E_v has a weight $\sum_{l=1}^{i} o_{vl}$ and a value $\sum_{q\in\mathcal{Q}} \lambda_{nvq} d_n \sum_{l=1}^{q} (o_{vl} - o_{vl+1}) \prod_{j=1}^{l} (1_{\{j\in\{1,2,...,i\}\}})$, where $1_{\{.\}}$ is the indicator function, i.e. it is equal to 1 if the condition in the subscript is true; otherwise it is zero, and $o_{vl+1} = 0$ for $l = q$.

Each maximum-value solution to the MCK instance can be mapped to a solution to the P_n instance of the same value as follows: For each item i in class E_v packed in the knapsack, place the i first layers of video v to the cache-node n. Clearly, the obtained solution stores no more data than the cache capacity.

Conversely, for every feasible solution to the P_n problem there is a feasible solution to the MCK instance of the same value. That is, for each sequence of i layers of video v placed in the cache-node n, we pack the item i of class v in the knapsack. Clearly, the obtained solution packs no more item weight than the knapsack capacity, and at most one item from each class is packed in the knapsack. □

[D. Proof of Theorem 4.4]

In order to prove Theorem 4.4, we first present the following lemma, which is proved in [122]: 4 For any set of arbitrary positive numbers $p_1, p_2, ..., p_T$, $w_1, w_2, ..., w_T$, $T \in \mathcal{Z}^+$, if $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq, ..., \geq \frac{p_T}{w_T}$, then $\sum_{i=1}^{j} p_i \geq \frac{\sum_{i=1}^{j} w_i}{\sum_{i=j+1}^{T} w_i} \sum_{i=j+1}^{T} p_i$, $\forall j \in \{1, 2, ..., T-1\}$.

Then, we note that at stage 2 of the LCC algorithm, the cache-nodes are asked to optimize their *local* demand, given that a portion of their cache space is already occupied by *globally* popular content (stage 1). Clearly, the latter constrains the optimization that takes place at each node, while it may introduce a loss to the local demand objective. The following lemma provides a bound to this loss.

Lemma 5. Let P_n^* and $P_n^*(\mathcal{A}_n)$ be the optimal solution values of the problems P_n and $P_n(\mathcal{A}_n)$ respectively. Then, $P_n^*(\mathcal{A}_n) \geq (1 - \frac{|\mathcal{A}_n|+s}{S_n}) \cdot P_n^*$

Proof. For a video v placed in the cache n, a sequence of layers $\{1, 2, ..., i\}$ will be cached. We define the weight $w_v = \sum_{l=1}^{i} o_{vl}$ and the value $p_v = \sum_{q\in\mathcal{Q}} \lambda_{nvq} d_n \sum_{l=1}^{q} (o_{vl} - o_{vl+1}) \prod_{j=1}^{l} (1_{\{j\in\{1,2,...,i\}\}})$, where $o_{vl+1} = 0$ for $l = q$. Here, w_v and p_v capture the cache space occupied by video v and the delay savings respectively.

We denote with $\mathcal{V}_n = \{1, 2, ..., T\}$ the set of videos placed in the cache of node n according to the P_n^* solution in descending order of their $\frac{p_v}{w_v}$ values. Then, we find an element $j \in \mathcal{V}_n$ such that:

$$S_n - |\mathcal{A}_n| - s \leq \sum_{v=1}^{j} w_v \leq S_n - |\mathcal{A}_n| \tag{16}$$

We also define the sets $\Gamma_j = \{1, ..., j\}$ and $\Delta_j = \{j+1, ..., T\}$. We can show that:

$$P_n^*(\mathcal{A}_n) \geq \sum_{v \in \Gamma_j} p_v \tag{17}$$

This is because the total size of the videos in Γ_j is less or equal to $S_n - |\mathcal{A}_n|$ and $P_n(\mathcal{A}_n)$ is the optimal solution value for node n when the available cache space of n is $S_n - |\mathcal{A}_n|$. Then, we show that:

$$\begin{aligned} \sum_{v \in \Gamma_j} p_v &\geq \frac{\sum_{v \in \Gamma_j} w_v}{\sum_{v \in \Delta_j} w_v} \cdot \sum_{v \in \Delta_j} p_v \\ &\geq \frac{S_n - |\mathcal{A}_n| - s}{|\mathcal{A}_n| + s} \cdot \sum_{v \in \Delta_j} p_v = (\frac{S_n}{|\mathcal{A}_n| + s} - 1) \cdot \sum_{v \in \Delta_j} p_v \end{aligned} \tag{18}$$

where the first inequality is because of lemma 7. The second inequality is because of inequality (16) and the fact that $\sum_{v \in \Delta_j} w_v = S_n - \sum_{v \in \Gamma_j} w_v$.

For any positive constant c it holds that: if $y \geq x \geq 0$, then $\frac{y}{y+c} \geq \frac{x}{x+c}$ [122]. Hence, replacing with $y = P_n^*(\mathcal{A}_n)$, $x = (\frac{S_n}{|\mathcal{A}_n|+s} - 1) \cdot \sum_{v \in \Delta_j} p_v$ and $c = \sum_{v \in \Delta_j} p_v$ and using inequalities (17) and (18), we obtain that:

$$\begin{aligned} \frac{P_n^*(\mathcal{A}_n)}{P_n^*(\mathcal{A}_n) + \sum_{v \in \Delta_j} p_v} &\geq \frac{(\frac{S_n}{|\mathcal{A}_n|+s} - 1) \cdot \sum_{v \in \Delta_j} p_v}{(\frac{S_n}{|\mathcal{A}_n|+s} - 1) \cdot \sum_{v \in \Delta_j} p_v + \sum_{v \in \Delta_j} p_v} \\ &= \frac{\frac{S_n}{|\mathcal{A}_n|+s} - 1}{\frac{S_n}{|\mathcal{A}_n|+s}} = 1 - \frac{|\mathcal{A}_n| + s}{S_n} \end{aligned} \tag{19}$$

Finally, we have:

$$\begin{aligned} \frac{P_n^*(\mathcal{A}_n)}{P_n^*} &= \frac{P_n^*(\mathcal{A}_n)}{\sum_{v \in \Gamma_j} p_v + \sum_{v \in \Delta_j} p_v} \\ &\overset{(17)}{\geq} \frac{P_n^*(\mathcal{A}_n)}{P_n^*(\mathcal{A}_n) + \sum_{v \in \Delta_j} p_v} \\ &\overset{(19)}{\geq} 1 - \frac{|\mathcal{A}_n| + s}{S_n} \end{aligned} \tag{20}$$

where the first equality holds by the definition of P_n^*. □

Lemma 5 serves as a building block for bounding the overall performance of LCC algorithm, and therefore it facilitates the derivation of Theorem 4.4. To show this, we start by denoting with S^{LCC} and S^{OPT} the delay savings achieved by the LCC and the optimal solution to the R_m problem respectively. Then, we divide S^{LCC} into two parts; (i) S_l^{LCC} that captures the delay savings incurred when user requests are served by their *local* cache node instead of another cache-node, and (ii) S_g^{LCC} that stands for the additional delay savings incurred when requests are served by any of the cache nodes instead of a content server. Similarly, we introduce the values S_l^{OPT} and S_g^{OPT} for the optimal solution. Then, we prove that:

$$\begin{aligned} S_l^{LCC} &\geq \sum_{n\in\mathcal{N}_m} \frac{P_n^*(\mathcal{A}_n)}{d_n} \min_{n'\in\mathcal{N}_m\setminus n} d_{nn'} \\ &\geq \sum_{n\in\mathcal{N}_m} (1-F-\frac{2s}{S_n})\frac{P_n^*}{d_n} \min_{n'\in\mathcal{N}_m\setminus n} d_{nn'} \\ &\geq (1-F-\frac{2s}{\min\limits_{n\in\mathcal{N}_m} S_n}) \sum_{n\in\mathcal{N}_m} \frac{\min\limits_{n'\in\mathcal{N}_m\setminus n} d_{nn'}}{\max\limits_{n'\in\mathcal{N}_m\setminus n} d_{nn'}} \frac{P_n^*}{d_n} \max_{n'\in\mathcal{N}_m\setminus n} d_{nn'} \\ &\geq \rho'\mu' S_l^{OPT} \end{aligned} \tag{21}$$

where the first inequality is because on its right hand side we always consider the minimum possible delay savings per request, i.e., the case that the closest to n node has cached the requested layer. The second inequality is based on Lemma 5 and the fact that $|\mathcal{A}_n|$ in stage 2 of LCC algorithm is upper-bounded by $F\cdot S_n + s$, $\forall n \in \mathcal{N}_m$. The third inequality is obtained after simple algebra, and the last inequality is because we always consider the maximum possible delay savings per request on the left hand side. Similarly, we can show that:

$$S_g^{LCC} \geq \rho\cdot\mu\cdot S_g^{OPT} \tag{22}$$

where we have applied Lemma 5 for a single cache-node, indexed by $n=0$, of capacity $C_0 = \sum_{n\in\mathcal{N}_m} S_n$ and $|\mathcal{A}_0| = (1-F)\cdot\sum_{n\in\mathcal{N}_m} S_n$ (according to the stage 1 of LCC). By summing (21) and (22) we complete the proof of theorem 4.4.

Bibliography

[1] Cisco, Visual Networking Index: The Zettabyte Era—Trends and Analysis, May 2015, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html

[2] J. Wang. 1999. A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communication Review, vol. 29, no. 5, pp. 36-46.*

[3] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. 2012. A survey of information-centric networking. *IEEE Communications Magazine, vol. 50, no. 7, pp. 26-36.*

[4] V. Sourlas, L. Gkatzikis, P. Flegkas and L. Tassiulas. 2013. Distributed Cache Management in Information-Centric Networks. *IEEE Transactions on Network and Service Management, vol. 10, no. 3, pp. 286-299.*

[5] Hewlett Packard, "HP 3PAR StoreServ Storage", Primary Storage Architecture Products, http://www8.hp.com/uk/en/products/data-storage/3parstoreserv.html , Nov. 2014.

[6] Disk Drive Prices (1955-2014). See http://www.jcmit.com/diskprice.htm.

[7] Ericsson Media Delivery Network Universal Cache, http://www.ericsson.com.

[8] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "Cache in the Air: Exploiting Content Caching and Delivery Techniques for 5G Systems", *IEEE Communications Magazine, vol. 52, no. 2, 2014.*

[9] B. Kaufman and B. Aazhang. 2008. Cellular networks with an overlaid device to device network. Proc. IEEE Asilomar Conference on Signals, Systems and Computers.

[10] Mobile Europe, *Altobridge debuts intel-based network edge small cells caching solution*, 2013, http://www.mobileeurope.co.uk/Press-Wire/altobridge-debuts-intel-based-hierarchical-network-edge-caching-solution.

[11] Light Reading, *NSN Adds ChinaCache Smarts to Liquid Applications*, 2014, http://www.lightreading.com/mobile/4g-lte/-nsn-adds-chinacache-smarts-to-liquid-applications-/d/d-id/708280.

[12] Saguna, *Saguna Open-RAN*, 2015, http://www.saguna.net/news-events/press-releases/saguna-expands-open-ran-platform-bringing-cdns-content-caching-and-otts-together-in-the-mobile-radio-edge.

[13] Linksys, "Smart Wi-Fi routers", `http://www.linksys.com/en-us/smartwifi`.

[14] HiWiFi, `http://www.hiwifi.com/j2`.

[15] J. Erman, A. Gerber, M.T. Hajiaghayi, "To Cache or Not to Cache - The 3G Case", *IEEE Internet Computing, vol. 15, no. 2, pp. 27-34, 2011.*

[16] B.A. Ramanan, L.M. Drabeck, M. Haner, N. Nithi, T.E. Klein, C. Sawkar, "Cacheability Analysis of HTTP traffic in an Operational LTE Network", *Wireless Telecommunications Symposium*, 2013.

[17] J. Dai, Z. Hu, B. Li, J. Liu, and B. Li, "Collaborative Hierarchical Caching with Dynamic Request Routing for Massive Content Distribution", *IEEE Conference on Computer Communications (Infocom), pp. 2444-2452, 2012.*

[18] S. Borst, V. Gupta, and A.Walid, "Distributed Caching Algorithms for Content Distribution Network", *IEEE Conference on Computer Communications (Infocom), pp. 1-9, 2010.*

[19] K. Poularakis, L. Tassiulas, "On the complexity of Optimal Caching in Hierarchical Networks", *IEEE Transactions on Communications, 2015, subject to major revision.*

[20] J. G. Andrews, "Seven ways that hetnets are a cellular paradigm shift", *IEEE Communications Magazine, vol. 51, no. 3, pp. 136-144, 2013.*

[21] K. Poularakis, G. Iosifidis, L. Tassiulas, "Approximation Algorithms for Mobile Data Caching in Small Cell Networks", *IEEE Transactions on Communications, vol 62, no. 10, pp. 3665-3677, 2014.*

[22] K. Poularakis, G. Iosifidis, I. Pefkianakis, L. Tassiulas, Martin May, "Mobile Data Offloading through Caching in Residential 802.11 Wireless Networks", *IEEE Transactions on Network and Service Management, 2015, subject to major revision.*

[23] K. Poularakis, G. Iosifidis, A. Argyriou, I. Koutsopoulos, L. Tassiulas, "Caching and Operator Cooperation Policies for Layered Video Content Delivery", *IEEE International Conference on Computer Communications (INFOCOM), 2016, to appear.*

[24] K. Poularakis, G. Iosifidis, A. Argyriou, L. Tassiulas, "Video Delivery over Heterogeneous Cellular Networks: Optimizing Cost and Performance", *IEEE Conference on Computer Communications (Infocom), pp. 1078-1086, April 2014.*

[25] K. Poularakis, G. Iosifdis, V. Sourlas, L. Tassiulas, "Exploiting Caching and Multicast for 5G Wireless Networks", *IEEE Transactions on Wireless Communications, 2015, to appear.*

[26] K. Poularakis, L. Tassiulas, "Code, Cache and Deliver on the Move: A Novel Caching Paradigm in Small-cell Networks", *IEEE Transactions on Mobile Computing, 2015, to appear.*

[27] N. Laoutaris, V. Zissimopoulos and I. Stavrakakis, "On the optimization of storage capacity allocation for content distribution', *Computer Neworks, vol. 47, no. 3, pp. 409-428, 2005.*

[28] I. Baev, R. Rajaraman, C. Swamy, "Approximation Algorithms for Data Placement Problems. *SIAM Journal on Computing, vol.38, no.4, pp. 1411-1429*, 2008.

[29] M.R. Korupolu, C.G. Plaxton, R. Rajaraman, "Placement algorithms for hierarchical cooperative caching", *ACM-SIAM Symposium on Discrete Algorithms.*

[30] A. Leff, J. Wolf, and P. Yu, "Replication algorithms in a remote caching architecture", *IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 11, pp. 1185-1204*, 1993.

[31] K. Poularakis, L. Tassiulas, "Optimal Cooperative Content Placement Algorithms in Hierarchical Cache Topologies", *Proc. Annual Conference on Information Sciences and Systems*, 2012

[32] J. W. Jiang, S. Ioannidis, L. Massoulie, and F. Picconi, "Orchestrating Massively Distributed CDNs", *ACM International Conference on emerging Networking EXperiments and Technologies*, 2012.

[33] K. Shanmugam, N. Golrezaei, A.G. Dimakis, A. Molisch, G. Caire, "FemtoCaching: Wireless Content Delivery Through Distributed Caching Helpers", *IEEE Transactions on Information Theory, vol. 59, no. 12, pp. 8402-8413*, 2013.

[34] M. M. Amble, P. Parag, S. Shakkottai, and L. Ying, "Content-Aware Caching and Traffic Management in Content Distribution Networks", in Proc. IEEE Infocom, pp. 2858-2866, 2011.

[35] M. Dehghan, A. Seetharam, B. Jiang, T. He, T. Salonidis, J. Kurose, D. Towsley and R. Sitaraman, "On the Complexity of Optimal Routing and Content Caching in

Heterogeneous Networks", *IEEE International Conference on Computer Communications*, 2015.

[36] J. Hachem, N. Karamchandani, S. Diggavi, "Content Caching and Delivery over Heterogeneous Wireless Networks", *IEEE International Conference on Computer Communications*, 2015.

[37] F. Pantisano, M. Bennis, W. Saad, and M. Debbah, "In-Network Caching and Content Placement in Cooperative Small Cell Networks", International Conference on 5G for Ubiquitous Connectivity (5GU), pp. 128-133, 2014.

[38] K. Hamidouche, W. Saad and M. Debbah, "Many-to-Many Matching Games for Proactive Social-Caching in Wireless Small Cell Networks", *International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, 2014.

[39] P. Ostovari, A. Khreishah, J. Wu, "Cache Content Placement Using Triangular Network Coding", *Proc. IEEE Wireless Communications and Networking Conference*, 2013.

[40] A. Kumar and W. Saad, "On the Tradeoff between Energy Harvesting and Caching in Wireless Networks", IEEE International Conference on Communication Workshop (ICCW), pp. 1976-1981, 2015.

[41] B. Blaszczyszyn and A. Giovanidis, "Optimal geographic caching in cellular networks", IEEE International Conference on Communications, June 2015.

[42] J. Dai, F. Liu, B. Li, B. Li, and J. Liu, "Collaborative caching in wireless video streaming through resource auctions", *IEEE Journal on Selected Areas in Communications, vol. 30, no. 2, pp. 458-466*, 2012.

[43] J. Yue, B. Yang, C. Chen, X. Guan, W Zhang, "Femtocaching in video content delivery: Assignment of video clips to serve dynamic mobile users", *Computer Communications, vol. 51, pp. 60-69*, 2014.

[44] A. Liu and V. K. N. Lau. 2013. Mixed-timescale precoding and cache control in cached MIMO interference network. *IEEE Transactions on Signal Processing, vol. 61, no. 24, pp. 6320-6332.*

[45] B. Zhou, Y. Cui and M. Tao, "Optimal Dynamic Multicast Scheduling for Cache-Enabled Content-Centric Wireless Networks", IEEE International Symposium on Information Theory (ISIT), pp. 1412-1416, 2015.

[46] C. Su and L. Tassiulas, "Joint broadcast scheduling and user's cache management for efficient information delivery", *Wireless Networks, vol. 6, no. 4, pp 279-288*, 2000.

[47] MA. Maddah-Ali and U. Niesen. "Fundamental Limits of Caching", *IEEE International Symposium on Information Theory*, 2013.

[48] V.A. Siris, X. Vasilakos, G. C. Polyzos, "Efficient Proactive Caching for Supporting Seamless Mobility", *arXiv:1404.4754*, 2014.

[49] Y. Guan, Y. Xiao, H. Feng, C-C. Shen, L. J. Cimini Jr, "MobiCacher: Mobility-Aware Content Caching in Small-Cell Networks", *IEEE Global Communications Conference*, 2014.

[50] Miroslav Chlebík and Janka Chlebíková, "Inapproximability Results for Bounded Variants of Optimization Problems", *Fundamentals of Computation Theory*, vol. 2751, pp. 27-38, 2003.

[51] G. L. Nemhauser, L. A. Wolsey, "Best Algorithms for Approximating the Maximum of a Submodular Set Function", *Mathematics of Operations Research*, vol. 3, no. 3, 1978.

[52] A. Schrijver, "Combinatorial optimization: polyhedra and efficiency", *Springer*, vol. 24, 2003.

[53] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5g wireless networks", *IEEE Communications Magazine, vol. 52, no. 8, pp. 82-89, 2014.*

[54] E. Bastug, M. Bennis, and M. Debbah, "Anticipatory caching in small cell networks: A transfer learning approach", *1st KuVS Workshop on Anticipatory Networks, 2014.*

[55] P. Blasco and D. Gunduz, "Learning-Based Optimization of Cache Content in a Small Cell Base Station", *IEEE International Conference on Communications, pp. 1897-1903, 2014.*

[56] D. Bertsimas and J. N. Tsitsiklis, "Introduction to Linear Optimization", *Belmont, MA: Athena Science*, 1997.

[57] Mosek Optimization Software, [online] http://www.mosek.com

[58] A. J. Hoffman and J. B. Kruskal, "Integral boundary points of convex polyhedra", *in 50 Years of Integer Programming 1958-2008, Springer*, pp. 49-76, 2010.

[59] Michael J. Quinn, "Parallel computing (2nd ed.): theory and practice", *McGraw-Hill, Inc.*, New York, NY, 1994.

[60] P. Goundan and A. Schulz, "Revisiting the greedy approach to submodular set function maximization", preprint 2009, available at http://www.optimization-online.org.

[61] M. Hefeeda and O. Saleh, "Traffic Modeling and Proportional Partial Caching for Peer-to-Peer Systems", *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1447-1460, 2008.

[62] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube traffc characterization: A view from the edge", *in Proc. ACM SIGCOMM*, pp. 15-28, 2007.

[63] A. Ghosh, J. Zhang, J. Andrews, and R. Muhamed, "Fundamentals of LTE", *Prentice Hall Communications Engineering and Emerging Technologies Series*, 2010

[64] D. B. Shmoys, E. Tardos, and K. Aardal, "Approximation Algorithms for Facility Location Problems", in Proc. of ACM STOC, 1997.

[65] M. Korupolu, C. Plaxton, R. Rajaraman, "Analysis of a Local Search Heuristic for Facility Location Problems", in Proc. of SODA, 1998

[66] J. Zhang, B. Chen, and Y. Ye, "A Multi-Exchange Local Search Algorithm for the Capacitated Facility Location Problem", Mathematics of Operations Research, vol. 30, no. 2, 2005.

[67] A. Aggarwal, L. Anand, M. Bansal, N. Garg, N. Gupta, S. Gupta, S. Jain, "A 3-Approximation for Facility Location with Uniform Capacities", in Proc. of IPCO, 2010.

[68] M. Bateni, and M. Hajiaghayi, "Assignment Problem in Content Distribution Networks: Unsplittable Hard-Capacitated Facility Location", ACM Transactions on Algorithms, vol. 8, no. 3, 2012.

[69] B. Behsaz, M. R. Salavatipour, Z. Svitkina "New Approximation Algorithms for the Unsplittable Capacitated Facility Location Problem", in Proc. of SWAT, 2012.

[70] D. Astely, E. Dahlman, A. Furuskar, Y. Jading, M. Lindstrom, and S. Parkvall, "LTE: The Evolution of Mobile Broadband", *IEEE Communications Magazine*, vol. 47, no. 4, 2009.

[71] Y. Bartal, "Probabilistic approximations of metric spaces and its algorithmic applications", in IEEE Symposium on Foundations of Computer Science, 1996, pp. 184-193

[72] C. Courcoubetis, R.R. Weber, "Pricing Communication Networks:Economics, Technology and Modelling", Wiley Europe, 2003.

[73] N. Choi, K. Guan, D. C. Kilper, and G. Atkinson, "In-network caching effect on optimal energy consumption in content centric networking", IEEE International Conference on Communications (ICC), pp. 2889-2894, June 2012.

[74] Google Drive, "Buy and manage storage plans", 2015, www.support.google.com/drive.

[75] M. El-Sayed, A. Mukhopadhyay, C. Urrutia-Valdes, and Z.J. Zhao, "Mobile Data Explosion: Monetizing the Opportunity Through Dynamic Policies and QoS Pipes", Bell Labs Technical Journal, vol. 16, no. 2, pp. 79-100, 2011.

[76] C. Joe-Wong, S. Seny, S. Ha, "Offering Supplementary Wireless Technologies: Adoption Behavior and Offloading Benefits", IEEE Conference on Computer Communications (Infocom), pp. 1061-1069, April 2013.

[77] GIGAOM, October 2011, https://gigaom.com/2011/10/04/2013-the-year-mobile-data-stops-being-profitable

[78] H. Schwartz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard", *IEEE Trans. on Circ. and Sys. for Video Tech.*, vol. 17, no. 9, 2007.

[79] F. Hartanto, J. Kangasharju, M. Reisslein, K. W. Ross, "Caching video objects: layers vs versions?", *Multimedia Tools Appl.*, vol. 31, no. 2, 2006.

[80] D. P. Bertsekas, R. Gallager, "Data Networks", Athena Scientific, 2003.

[81] W. Jiang, R. Zhang-Shen, J. Rexford and M. Chiang, "Cooperative content distribution and traffic engineering in an ISP network", *in Proc. of ACM SIGMETRICS*, 2009

[82] D. Bertsekas, A. Nedic, and A. Ozdaglar, "Convex Analysis and Optimization", *Athena Scientific Press*, 2003.

[83] T. Bektas, et al., "Exact Algorithms for the Joint Object Placement and Request Routing Problem in Content Distribution Networks", *Comp. and OR*, vol. 35, no. 12, 2008.

[84] K. Son, H. Kim, Y. Yi, and B. Krishnamachari, "Base Station Operation and User Association Mechanisms for Energy-Delay Tradeoffs in Green Cellular Networks", *IEEE JSAC*, vol. 29, no. 8, 2011.

[85] Video Trace Library: http://trace.eas.asu.edu.

[86] L. Peterson, B. Davie, R. van Brandenburg, "Framework for Content Distribution Network Interconnection (CDNI)", *IETF*, 2014, https://tools.ietf.org/html/rfc7336

[87] M.S. Bansal, V.C. Venkaiah, "Improved Fully Polynomial time Approximation Scheme for the 0-1 Multiple-choice Knapsack Problem", in Proc. of SIAM Conference on Discrete Mathematics, 2004.

[88] OFweek, 'China Telecom successfully deployed LTE eMBMS", June 2014, http://global.ofweek.com/news/China-Telecom-successfully-deployed-LTE-eMBMS-13100.

[89] Alcatel-Lucent, "eMBMS for More Efficient Use of Spectrum", November 2011, http://www2.alcatel-lucent.com/techzine/embms-for-more-efficient-use-of-spectrum.

[90] 3GPP releases, http://www.3gpp.org/specifications/releases/71-release-9.

[91] Ericsson, http://www.ericsson.com/res/thecompany/docs/press/backgrounders/lte-broadcast-press-backgrounder.pdf

[92] J. Erman, K.K. Ramakrishnan, "Understanding the super-sized traffic of the super bowl", ACM IMC, pp. 353-360, October 2013.

[93] M.Z. Shafiq, L. Ji, A.X. Liu, J. Pang, S. Venkataraman, J. Wang, "A First Look at Cellular Network Performance during Crowded Events", ACM SIGMETRICS, pp. 17-28, June 2013.

[94] V. Tokekar, A. K. Ramani, and S. Tokekar, "Analysis of Batching Policy in View of User Reneging in VoD System", IEEE Indicon, pp. 399-403, December 2005.

[95] M. Garey, D. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W. Freeman & Comp., San Francisco, 1979.

[96] S.Tombaz, P. Monti, K.Wang, A. Vastberg, M. Forzati, J. Zander, "Impact of Backhauling Power Consumption on the Deployment of Heterogeneous Mobile Networks", IEEE Global Communications Conference (Globecom), pp. 1-5, December 2011.

[97] A. Damnjanovic, J. Montojo, Y. Wei, T. Ji, T. Luo, M. Vajapeyam, T. Yoo, O. Song, and D. Malladi, "A survey on 3GPP heterogeneous networks", IEEE Transactions on Wireless Communications, vol. 18, no. 3, pp. 10-21, June 2011.

[98] B. Prabhakar, E. Uysal-Biyikoglu and A. El Gamal, "Energy-efficient transmission over a wireless link via lazy packet scheduling", IEEE Conference on Computer Communications (Infocom), pp. 386-394, April 2001.

[99] C. Peng, S. Lee, S. Lu, H. Luo, H. Li, "Traffic-Driven Power Saving in Operational 3G Cellular Networks", ACM International Conference on Mobile Computing and Networking (Mobicom), pp. 121-132, September 2011.

[100] N. Choi, K. Guan, D. C. Kilper, and G. Atkinson, "In-network caching effect on optimal energy consumption in content-centric networking", IEEE International Conference on Communications, pp. 2889-2894, June 2012.

[101] A. Patro, S. Govindan, S. Banerjee, "Observing Home Wireless Experience through WiFi APs", ACM International Conference on Mobile Computing and Networking (Mobicom), pp. 339-350, September 2013.

[102] K. Lee, J. Lee, Y. Yi, I. Rhee, S. Chong, "Mobile data offloading: how much can WiFi deliver?", IEEE/ACM Transactions on Networking, vol. 21, no. 2, pp. 536-551, April 2013.

[103] A. Hayrapetyan, D. Kempe, M. Pál, Z. Svitkina, "Unbalanced graph cuts", European Symposium on Algorithms (ESA), pp. 191-202, October 2005.

[104] Y. Zhu, Z. Zhang, Z. Marzi, C. Nelson, U. Madhow, B.Y. Zhao, H. Zheng, "Demystifying 60GHz Outdoor Picocells", ACM International Conference on Mobile Computing and Networking (MobiCom), pp. 5-16, 2014.

[105] Qualcomm, "Enabling Hyper-Dense Small Cell Deployments with UltraSON", https://www.qualcomm.com/documents/enabling-hyper-dense-small-cell-deployments-ultrason

[106] JPL's Wireless Communication Reference Website, Cell Sizes, http://www.wirelesscommunication.nl/reference/chaptr04/cellplan/cellsize.htm

[107] S. Gambs, M. Killijian, M. Cortez, "Next Place Prediction using Mobility Markov Chains", Workshop on Measurement, Privacy, and Mobility, pp. 3:1-3:6, 2012.

[108] D. Leong, A. G. Dimakis, and T. Ho, "Distributed storage allocations", IEEE Transactions on Information Theory, vol. 58, no. 7, pp. 4733-4752, 2012.

[109] V. Ntranos, G. Caire, A. Dimakis, "Allocations for Heterogenous Distributed Storage", IEEE International Symposium on Information Theory (ISIT), pp. 2761-2765, 2012.

[110] CPLEX: Linear Programming Solver. [Online]. Available: http://www.ilog.com.

[111] D. Ashbrook, T. Starner, "Learning Significant Locations and Predicting User Movement with GPS", International Symposium on Wearable Computers (ISWC), pp. 101-108, 2002.

[112] A. J. Nicholson and B. D. Noble, "Breadcrumbs: Forecasting mobile connectivity", ACM International Conference on Mobile Computing and Networking (MobiCom), pp. 46-57, 2008.

[113] J. Pang, B. Greenstein, M. Kaminsky, D. McCoy, and S. Seshan, "Wifi- reports: Improving wireless network selection with collaboration", IEEE Transactions on Mobile Computing, vol. 9, no. 12, pp. 1713-1731, 2010.

[114] K. Chung, H. Lam, Z. Liu, M. Mitzenmacher, "Chernoff-Hoeffding Bounds for Markov Chains: Generalized and Simplified", International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 124-135, 2012.

[115] G. Dantzig, "Discrete-Variable Extremum Problems", Operations Research Vol. 5, No. 2, pp. 266-288, 1957.

[116] 3GPP Specification, "Access Network Discovery and Selection Function Management Object", http://www.3gpp.org/dynareport/24312.htm

[117] P. Xia, Jo Han-Shin, J.G. Andrews, "Fundamentals of Inter-Cell Overhead Signaling in Heterogeneous Cellular Networks", IEEE Journal of Selected Topics in Signal Processing, pp. 257-269, 2011.

[118] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet", IEEE MultiMedia, vol. 18, no. 4, pp. 62-67, 2011.

[119] M. McNett and G. M. Voelker, "Access and mobility of wireless pda users", ACM SIGMOBILE Mobile Computing and Communications Review, vol. 9, no. 2, pp. 40-55, Apr. 2005.

[120] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Watch global, cache local: YouTube network traffic at a campus network-measurements and implications", SPIE/ACM Multimedia Computing and Networking Conference (MMCN), 2008.

[121] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014–2019, White Paper, February 2015.

[122] Y. Lien, "Some Properties of 0-1 Knapsack Problems", in Proc. of Conference on Combinatorics and Complexity, 1987.