Wireless networks QoS optimization using Coded Caching and machine learning algorithms

by

© *Mohamed Khalil*

A thesis submitted to the School of Graduate Studies

in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering,

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

Febrauary 2023

St. John's Newfoundland

Faculty of Engineering & Applied Science

Letter of Transmittal

I, Mohammed Amir, certify that my PhD thesis is prepared in accordance with the Faculty of Engineering and Applied Science guidelines and I am fully responsible for its contents. I am ready to proceed to have my thesis evaluated by my supervisory committee.

Mr. Mohamed Khalil

Date: 17 May 2022

Abstract

Proactive caching shows great potential to minimize peak traffic rates by storing popular data, in advance, at different nodes in the network. We study three new angles of proactive caching that were not covered before in the literature. We develop more practical algorithms that bring proactive caching closer to practical wireless networks.

The first angle is where the popularities of the cached files are changing over time and the file delivery is asynchronous. We provide an algorithm that minimizes files' delivery rate under this setting. We show that we can use the file delivery messages to proactively and constantly update the receiver finite caches. We show that this mechanism reduces the downloaded traffic of the network. The proposed scheme uses index coding [1], and app. A to jointly encodes the delivery of different demanded files with the cache updates to other receivers to follow the changes in the file popularities. An offline and online (dynamic) versions of the scheme are proposed, where the offline version requires knowledge of the file popularities across the whole transmission period in advance and the online one requires the file popularities for one succeeding time slot only. The optimal caching for both the offline and online schemes is obtained numerically.

The second angle is the study of segmented caching for delay minimization in networks with congested backhaul. Studies have mainly focused on proactively storing popular whole files. For certain categories of files like videos, this is not the best strategy. As videos can be segmented, sending later segments of videos can be less time-critical. Video is expected to constitute 82% of internet traffic by 2020 [2]. We study the effect of segmenting video caching decisions under the assumption that the backhaul is congested. We provide an algorithm for proactive segmented caching that optimizes the choice of segments to be cached to minimize delay and compare the performance to the whole file proactive caching.

The third angle focuses on using reinforcement learning for coded caching in networks with changing file popularities. For such a dynamic environment, reinforcement learning has the flexibility to learn the environment and adapt accordingly. We develop a reinforcement learning-based coded caching algorithm and compare its performance to rule-based coded caching.

Acknowledgements

I would like to express my appreciation to my supervisors, for supervising me and for their contribution and suggestions to this work. I would like to acknowledge the financial support provided by my supervisor, the Faculty of Engineering and Applied Science, the School of Graduate Studies, and the Natural Science and Engineering Research Council of Canada.

Contents

List of Tables

List of Figures

List of Abbreviations and Acronyms

Abbreviation\ Acronym	Description
BRNN	Bidirectional Deep Recurrent Neural Network
BS	Base Station
CA	Computation Assistant
CMAB	Combinatorical Multi Armed Bandit problem
D2D	Device to Device
LDC	Linear Dynamic Caching algorithm
LRU	Last Recently Used file Caching algorithm
MAB	Multi Armed Bandit problem
MU	Mobile Unit
OCC	Offline Coded Caching algorithm
QoS	Quality of service
RL	Reinforcement Learning
RLICCU	Reinforcement Learning Coded Caching-Updates

Abbreviation\ Acronym	Description
SRV	Server
UAV	Unmanned Ariel Vehicle

List of Symbols

Symbol	Description
$\mathcal{E}$	Calligraphy fonts used to denote set $\mathcal{E}$.
$\text{card}(\mathcal{S})$	Denotes the cardinally of the set $\mathcal{E}$.
$\binom{\mathcal{E}}{k}$	Denotes the set of all subsets of $\mathcal{E}$ with size k
$\oplus$	Denotes bit-wise XOR operation
$A \backslash B$	Represents the bits in A that are not in B.
$O(.)$	Denotes the worst case complexity order.
Φ	Denotes an empty set.
F_i	The ith file in the library stored at the BS.
$\mathcal{F}$	The library of L files stored at the BS.
$F_{i,m}$	The mth part of the ith file.
$\mathbf{d}_j$	The jth demand at the jth time slot.
$\mathbf{D}_j$	Vector of the last j demands.
$P_{i,t}$	The demand probability of file ith file at time slot t.
P_{D_K}	The aggregate demand probability of the demand vector $\mathbf{D}_K$.

Symbol	Description
S_i^{Φ}	The normalized size of the parts of the ith file that are cached at the placement phase.
$S_i^{\mathbf{D}_l}$	The normalized size of the parts of the ith file that are cached at each MU after D_l demands have been satisfied by the BS.
$S_{i,t}$	The normalized size of the parts of file ith file that are cached at time-slot t
$\boldsymbol{\mathcal{S}}_{\mathcal{F}}$	The set of all file parts $S_i, S_i^{D_1}, ..., S_K^{D_K}$.
M_{joint}	The delivery plus update message sent by the BS to the MUs.
M_{delivery}	The delivery part of the message M_{DU}.
M_{update}	The update part of the message M_{DU}.
R_t	The expected delivery rate can at time-slot t
R_d	The total expected delivery rate that is optimized offline.
R_{on}^t	The expected delivery rate at time slot t that is optimized online.
$U_{i,t}$	The size update of file ith file at time slot .
$\mathcal{U}^+$	The set of positive size updates.
$\mathcal{U}^-$	The set of negative size updates.
$\boldsymbol{b}_{g,t}$	Feature vector for time i, it contains weights that define the relative edge of each file for a specific g.
F_{minmax}	The minmax fairness measure

Table 1: Summary of the symbols used through out the paper.

List of publications

Journal Articles

1. M. Amir, E. Bedeer, M. H. Ahmed and T. Khattab, ”Coded Caching for Time-Varying Files Popularities and Asynchronous Delivery,” in *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1458-1472, 2021, doi: 10.1109/OJCOMS.2021.3091636.

2. M. Amir, E. Bedeer, Telex M. N. Ngatched, and T. Khattab, “Dynamic Coded Caching for Files with Time-Varying Features and Content,” *IEEE Trans. commun. theory*, Oct. 2022.

Conference papers

3. M. Amir, E. Bedeer, M. H. Ahmed, and T. Khattab, “Joint coding for proactive caching with changing file popularities,” in *Proc. IEEE 28th Annual International Symposium on Personal*, Indoor, and Mobile Radio Communications (PIMRC), Oct. 2017, pp. 1–6.

Under review

4. M. Amir, E. Bedeer, Telex M. N. Ngatched, and T. Khattab, “Dynamic Coded Caching for Files with Time-Varying Features and Content,” under the third round of review for possible publication in the *IEEE Trans. commun. theory.*

To be submitted soon

5. M. Amir, E. Bedeer, Telex M. N. Ngatched, M. H. Ahmed, and T. Khattab, “Learning-based Proactive Coded Caching with Changing File Popularities,” in preparation.

Chapter 1

Introduction

In this Chapter, we discuss the research motivation and objective. At the end of the Chapter, we provide the outline of the thesis. Then, in the following Chapters, we discuss how our proposed research provides a resolution for these areas.

1.1 Research Motivation

Cellular network traffic has shifted over the past decade from mainly locally generated instantaneous traffic (voice calls) to centrally generated delay-tolerant bulks of traffic (data-driven communication) [3]. For example, video constituted 82% of the internet traffic in 2020 [2]. This necessitates the development of new information-theoretic analysis and new algorithms for communication networks. Contrary to voice calls, the delay between the time of data generation and the time of data demand at the receivers is generally large. In this context, the data can be stored midway in different nodes along the paths from the transmitter to the receivers. Moreover, the shift from voice to data traffic contradicts the classical assumption of network analysis that the messages are generally independent. This assumption is not accurate for data communication where a piece of data is generally demanded by multiple users over time. Proactive caching is an efficient technique to reduce the peak traffic rate and the delivery sum rate. This is achieved by storing parts of the popular content, at various nodes in the networks, before being demanded by the mobile units (MUs). In a practical sense, proactive caching can minimize the total cost of transmission, as transmitters can optimize the time of performing proactive caching to be during less congested times (e.g overnight) where the resources are in abundance.

The proactive caching model is a two-phase communication model. During the first (placement) phase, a number of independently generated messages, each corresponding to one piece of content, are stored proactively at the transmitter and\or the receivers. The receivers' demands are assumed to be limited to a number of these messages. During the second (delivery)phase, the receivers' exact demands arrive at the transmitter and are satisfied in a manner that mainly reduces the transmission rate over this phase. The main difference compared to traditional communication is that the transmitter and receiver need to optimize the channel use for the two phases jointly rather than a single delivery phase, where the demand is probabilistic for the first phase.

As the research in the field of proactive caching is relatively new, there are still several problems that need deeper analysis and need the development of adequate caching algorithms. In this thesis, we focus on three main caching problems, asynchronous delivery coded caching for networks with changing file popularities, delay optimization of cache-aided networks with congested back-hauls, and reinforcement learning for coded caching.

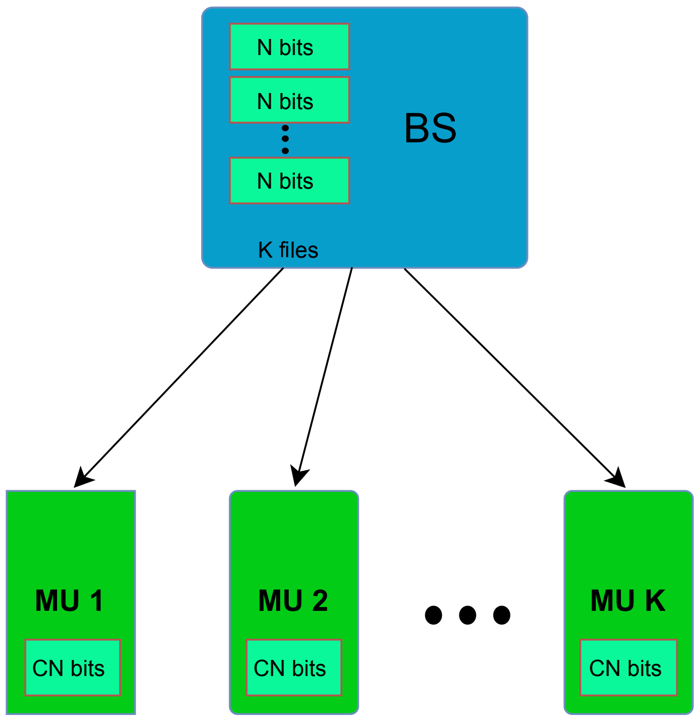


Figure 1.1: Network architecture.

1.1.1 Asynchronous coded delivery

Previous works found in the literature can be categorized into two groups in terms of delivery. The first group studied synchronous delivery with no change in file popularities paired with the use of coded [1] caching and/or delivery. The second group studied asynchronous delivery with changing file popularities but with uncoded caching and uncoded delivery. The use of coded caching and/or coded delivery with asynchronous delivery is not well explored in the literature, especially in the case of changing file popularity. However, asynchronous delivery is more dominant in practice, and a change in file popularities is expected over long delivery phases. Moreover, in most applications users expect their requests to be satisfied in a short time window. In such case their coded messages can not satisfy more than one user simultaneously. Coded messages however can be used to update other user caches. This can be beneficial in the case that the content server learns the files popularities and users preferences over time or in the case that file popularities changes over time. Applications with changing file popularity is a case not previously considered in a coded caching setting despite its prevalence in practical scenarios. An example is traffic information which change fast. Another is a weather app that stores information about daily weather changes. This brings a need for caching networks to be able to continuously update the mobile local caches and update their coding strategy to follow the change in the file popularities. We study coded caching when the file popularities are changing and the demands are asynchronous, and we develop algorithms that deliver users demands and update local caches.

[1]Throughout this work, coded caching refers mainly to index-coding different files/ parts of files together.

1.1.2 Reinforcement learning for asynchronous coded delivery

The use of machine learning techniques is a natural evolution for proactive caching. Machine learning has the flexibility to react to the changing dynamics of user requests and the complexity of optimizing the available resources. Coded caching has proved to be complex, where finding an optimal coding scheme is cumbersome, especially for a dynamic environment. Motivated by the previous, we develop a reinforcement learning-based coded caching algorithm for networks with changing file popularities and asynchronous delivery.

1.1.3 Delay analysis of caching in back-haul congested networks

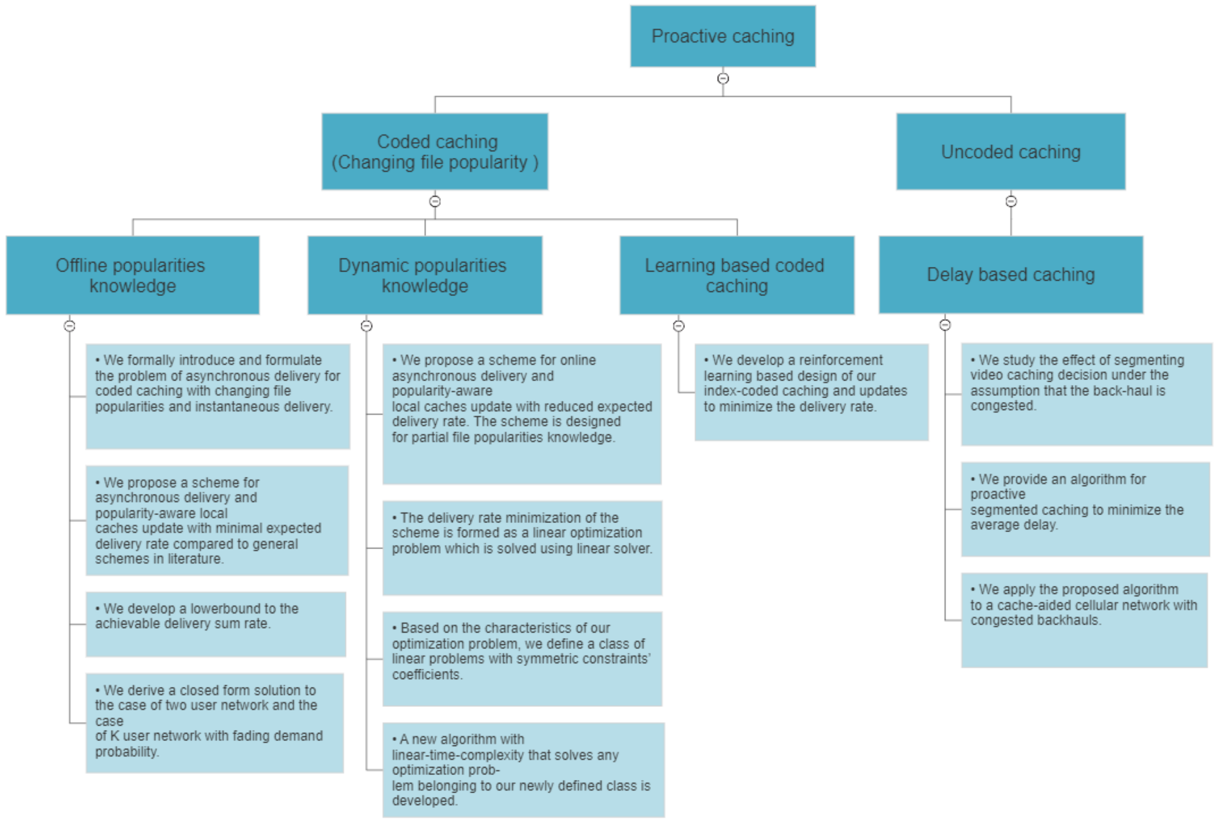
Proactive caching studies have mainly focused on proactively storing popular whole files. For certain categories of files like videos, this is not the best strategy. Storing popular files reduces the overall throughput. However, the delay relation to caching is more complicated and not well studied. In particular, queue delay is more related to the scheduling interaction between cached and uncached files. Also, as videos can be segmented, sending later segments of videos can be less time-critical. As such, studying the effect of segmenting video caching decisions under the assumption that the back-haul is congested is crucial to improve the average transmission delay. In particular, we study the effect of the segment's position in the file on the backhaul delay in comparison to segment's popularity and develop an algorithm to optimize the choice of cached segments to minimize transmission delay.

1.2 Research Objective and Contributions

The main objective of this thesis is to develop practical and efficient caching algorithms that help minimize peak data rate of wireless networks and that can work in dynamic environments with various degrees of knowledge about such environments. The main contributions of this thesis can be summarized as follows:

- We formally define the problem of asynchronous delivery for coded caching with changing file popularities.
- We provide an algorithm for asynchronous delivery and popularity-aware local cache update with a reduced expected delivery rate. The algorithm has two versions, one is built for complete prior knowledge of the file popularities changes and the other is for limited knowledge.
- We develop a new linear-time algorithm to solve the sum rate optimization problem as well as a similar class of linear optimization problems.
- We develop a reinforcement learning-based approach for designing the caching policy to minimize the delivery rate in a cache-aided network with changing file popularities.
- We develop a segmented caching algorithm for video delivery that minimizes user delay in cache-enabled congested back-haul wireless networks.

A detailed contribution chart is presented in Figure 1.3.


Proactive caching
Coded caching (Changing file popularity)
Uncoded caching
Offline popularities knowledge
Dynamic popularities knowledge
Learning based coded caching
Delay based caching

• We formally introduce and formulate the problem of asynchronous delivery for coded caching with changing file popularities and instantaneous delivery.

• We propose a scheme for asynchronous delivery and popularity-aware local caches update with minimal expected delivery rate compared to general schemes in literature.

• We develop a lowerbound to the achievable delivery sum rate.

• We derive a closed form solution to the case of two user network and the case of K user network with fading demand probability.

• We propose a scheme for online asynchronous delivery and popularity-aware local caches update with reduced expected delivery rate. The scheme is designed for partial file popularities knowledge.

• The delivery rate minimization of the scheme is formed as a linear optimization problem which is solved using linear solver.

• Based on the characteristics of our optimization problem, we define a class of linear problems with symmetric constraints' coefficients.

• A new algorithm with linear-time-complexity that solves any optimization problem belonging to our newly defined class is developed.

• We develop a reinforcement learning based design of our index-coded caching and updates to minimize the delivery rate.

• We study the effect of segmenting video caching decision under the assumption that the back-haul is congested.

• We provide an algorithm for proactive segmented caching to minimize the average delay.

• We apply the proposed algorithm to a cache-aided cellular network with congested backhauls.

Figure 1.2: Thesis summary

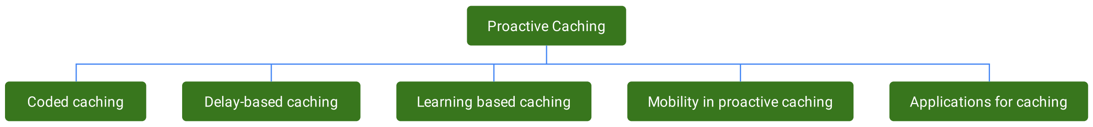


Figure 1.3: Literature review outline

1.3 Thesis Outline

In Chapter 2, we briefly discuss the related studies and highlight the areas not covered before. In the following Chapters 3, and 4, we study the proactive caching problem under time-Varying Files popularities and provide algorithms to optimize the transmission rate. In Chapter 3, we consider that the caching decisions are optimized offline with full prior knowledge about future file popularities variations, while in Chapter 4, we consider the case where caching decisions are dynamic based on limited knowledge of the file popularities variations. In Chapter 5, we provide a learning-based proactive caching algorithm. In Chapter 6, we study delay based caching. Finally, in Chapter 7, we provide our work conclusions and gained insights as well as future work.

1.4 Thesis summary

In this thesis, our main aim is to develop practical and efficient caching algorithms to work in real-world scenarios. In chapter three, we focus on the case where the file popularities of the library are caching over the delivery period. We studied the case when the information about the changing file popularities is available offline at the placement phase through external help. We develop a novel caching algorithm to continuously update the local caching by using indexing coding. The rate optimization of the proposed algorithm is formulated as an optimization that is solved using linear solvers. The achieved rate is compared to a developed genie-aided lower-bound and other caching algorithms in the literature.

In chapter 4, we study the same system model when the information about the changing file popularities is available only one slot ahead. The main aim of the chapter is to add flexibility and practicality to the main algorithm presented in the previous chapter. The algorithm is designed to react swiftly in situations where it is hard to accurately predict file popularities over a long period of time or a prediction mismatch is discovered. The achieved rate is compared to the main algorithm and the rate loss due to information limitations is discussed. To improve the practicality of the developed algorithm, even more, a new linear time algorithm to solve the underlying linear optimization problem is developed.

In chapter 5, we discuss the complexity of the algorithm provided in chapter three. Because of it large memory resource demands for algorithm design, we develop a dynamic programming approach to solve the underlying linear optimization which is less memory intensive at a cost of needing more time and being slightly less rate-efficient. In the second part of the chapter, we discuss the case where information about file popularities is completely unavailable and it is learned by the network over time. We develop a reinforcement learning-based algorithm for this scenario, where the Deep reinforcement learning actor-critic algorithm is used due to its power in learning in continuous state space.

In Chapter 6, we study the delay optimization of networks with congested backhauls using proactive caching. We show an interesting, yet counterintuitive, result that caching a chosen set of segments of less popular files is advantageous delay-wise in comparison to popularity based proactive caching. We present a reinforcement learning-based algorithm that optimizes the set segments to be cached in order to minimize the average delay.

While not discussed in depth, the algorithms provided in these chapters are rather generic and can be optimized to achieve different targets. Age of Information, the cache consistency problem, and mobility are direct applications of the provided work. Since the provided algorithms provide zero-rate updates to local caches, they can be easily used for cache updates to maintain consistency and/ or respond to user mobility-induced cache update requests.

Chapter 2

Literature Review

In this chapter, we briefly discuss the related studies and the areas not covered before. Then, in the following chapters, we discuss how our proposed research provides solutions for the problems founds in these areas.

The idea of proactive caching at the edge of wireless networks was introduced in [4], where the authors proposed the idea of small-cell network cloud as a technique to increase cloud capacity, relieve the backhaul constraints and increase the peak rate. The transmission characteristics of proactive caching were studied in [5], where the authors derived closed-form expressions for the outage probability and the average delivery rate as a function of the signal-to-interference-plus-noise ratio, small-cells density, transmission, bitrate, cache size, file length, and file popularity. The caching policy design is a challenging problem especially if wireless networks' realistic parameters are considered. The authors of [6] formulated the joint routing and caching problem aiming to maximize the fraction of content requests served locally by the deployed Small Base Stations (SBS)s under realistic wireless network constraints. The proactive caching research focuses on two main caching strategies. The first is storing raw files in respective caches [4–8]. The second is the coding of different files together (e.g., index coding [1]) to optimize the usage of the resources [9–27].

2.1 Coded caching

The authors of [9] presented a novel coded caching scheme that exploits the broadcast nature of the channel and index coding to further improve the performance of traditional proactive caching. They introduced an information-theoretic framework for the analysis of cache-aided communication in the context of broadcast channels and showed that the availability of caches in a broadcast setting provides a coded multi-casting gain. The main difference compared to traditional proactive caching is that the transmitter and the receiver need to optimize the channel use for the two phases jointly (the demand is probabilistic for the first phase) rather than for a single delivery phase. This work was extended to the case of non-uniform demands in [10–12]. The authors of [12] provided an optimization framework that handles various heterogeneous aspects of practical coded caching. The framework is used to develop a coded caching scheme that works for non-uniform file length, non-uniform file popularity, and non-uniform user cache size simultaneously. The authors of [13] developed an online version of the coded caching scheme proposed in [9]. In their model, users have a new different demand vector each time slot, and there is a new ”popular files” set. The scheme sends an index-coded message of the parts of the files demanded but that are missing from the users' caches. The users then replace their caches with index-coded random parts of the most recently sent files. The authors of [14] focused on the case of uncoded placement and coded delivery. Furthermore, some work developed decentralized schemes for the coded caching problem such as the works in [15, 16]. In [15], the authors proposed centralized and decentralized caching schemes with smaller delivery rates when the number of users is relatively large compared to the number of files. Coded caching was then extended to different wireless network models. The achievable tradeoff between cache size and download rate in decentralized caching systems was analyzed in [16]. Interference channel with coded caching was studied in [17–19]. In [17], the authors formally introduced the communication over a cache-aided interference channel where both transmitters and receivers are equipped with caches and provided an achievable scheme and computed its achievable degree-of-freedom (DoF). The DoF of the cache aide interference channel was then computed in [18] in terms of the cache size. Meanwhile, the authors of [19] studied the two-user Gaussian interference channel and computed the DoF for a separation strategy that divides the physical and network layers and showed that separation is optimal in regimes where the receiver caches are large. They provided a constant-factor approximation of the system's DoF. Coded caching within wireless device-to-device (D2D) network was studied in [20], [21]. The authors proposed decentralized caching schemes with smaller delivery rates when the number of users is relatively large compared to the number of files. Uncoded cache placement and decentralized coded caching in the finite file size regime was studied in [23], where the authors proposed a decentralized random coded caching scheme and a partially decentralized sequential coded caching scheme. They proposed a caching strategy where the users send coded messages to each other in order to collectively satisfy their demands and showed that a caching D2D wireless network achieves the same throughput scaling law of the infrastructure-based coded multicasting scheme if the number of users and number of files in the system is large. The authors of [24] propose game-theoretic proactive caching and study how the selfishness of different parties may influence the overall wireless proactive caching. They designed caching algorithms by considering the relations and interactions among different network nodes using game theory.

All the previous coded caching works assumed synchronous demand/delivery. The use of index-coded caching and/or index-coded delivery for asynchronous delivery is not well explored in the literature. Caching for asynchronous coded delivery was studied in [25], [26], where the authors studied the coded caching where the user's request is not instantaneously satisfied, instead, it is satisfied before a user set deadline for delivery. A general algorithm for asynchronous coded delivery delivery is yet to be developed. Cases like instantaneous response to requests and general request time overlaps are not fully studied in literature. Moreover, in most applications where users expect their requests to be satisfied in a short time window. In such case their coded messages can not satisfy more than one user simultaneously. Coded messages however can be used to update other user caches. This can be beneficial in the case that the content server learns the files popularities and users preferences over time or in the case that file popularities changes over time. Applications with changing file popularity is a case not previously considered in a coded caching setting despite its prevalence in practical scenarios. An example is traffic information which change fast. Another is a weather app that stores information about daily weather changes.

2.2 Delay-Based Caching

Proactive caching based on files popularities generally reduces delay as it stores data near end users and bypasses the congested resources of the network links. However, the method of storing the most popular content is not proven to be optimal. Towards that end, other works focused on delay-driven proactive caching [28–34]. Some of this work was developed for web proxy caching. The authors of [28] developed an algorithm that caches content based on both of its popularity and fetching time. They showed that their algorithm is more advantageous in reducing delay than popularity-driven algorithms like last recently used(LRU) algorithm. Segmented caching was studied in various works [e.g. [29, 30]], the authors of [29] provided a web proxy segmented caching algorithm that prioritizes earlier segments of videos. They showed that it is advantageous in terms of the bit-hit ratio and the fraction of requests that require a delayed start. They showed that segmentation-based caching is especially advantageous when the cache size is limited and files popularities changes over time. On the other hand, the authors of [30] focused on per-segment popularity and provided an algorithm that can cache individual file segments based on their popularity and their distance to the file start. In [31], the authors proposed a popularity-aware partial caching algorithm that minimizes the average initial delay of the system while allowing a small deviation from the desired starting point. The authors of [32] focused on cooperative cell caching for mobile networks, where each base station cache popular contents to improve the overall delay performance of users. Likewise, the authors of [33] focused on caching for mobile networks, but with D2D communication. They provided an algorithm that minimizes the average transmission delay for this type of network. The algorithm addresses a more general scenario, in which the values of system parameters potentially change over time. On the other hand, in [34], the authors investigated the trade-off between coded caching gain and delivery delay. They developed a computationally efficient caching scheme that effectively exploits coding opportunities while respecting delivery-delay constraints.

2.3 Learning-Based Caching

Using a learning algorithm to optimize the proactive caching performance was studied in [8, 35–47]. In [8], content caching is optimized through the use of a reinforcement learning algorithm while the transmitter is oblivious to the requests statistics. In [35], the authors proposed an online proactive caching scheme based on a bidirectional deep recurrent neural network (BRNN) model to predict time-series content requests and update edge caching accordingly. While the authors of [36] proposed a novel deep learning-based proactive caching framework in cellular networks, called DeepCachNet, in which a vast amount of data is collected from the mobile devices of users connected to small base stations. The deep-learning methods called auto-encoder and stacked denoising autoencoders are applied to the collected data to extract the features of users and content, respectively. The extracted features are then used to estimate the content popularity at the core network. Based on the estimated content popularity, the strategic content is cached at the small base stations to obtain higher backhaul offloading and user satisfaction. The authors of [37] proposed a proactive caching mechanism for mobile edge computing architecture to reduce transmission cost and improve user quality of experience. They exploited a transfer learning-based approach for estimating content popularity and then formulated the proactive caching optimization model. They developed a greedy algorithm for solving the cache content placement problem as the optimization problem is NP-hard. In [38], the authors studied the optimization of the cache content of small cells. They modeled the problem as a multi-armed bandit (MAB) problem where the cache content placement is optimized based on the demand history. The authors of [39] modeled the proactive caching problem as a Markov decision process, where the environment is the knowledge of the channel quality, the content profile, and the user-access behavior, and the target is to minimize the long-term average energy cost. They provide a threshold-based proactive caching scheme, which dynamically caches or removes content. The authors of [40] carried out a proactive caching experiment and collected users' mobile traffic data from a telecom operator. They used big data platform to locally analyze data and make the caching decision. Their results showed that proactive caching gains depending on the level of available information and storage size. The work in [41] leverages the spatial and social structure of the network to predict user demands. They proposed an algorithm that predicts the set of influential users to (proactively) cache important contents and distribute them to their social ties via D2D communications. The authors of [39] used reinforcement learning-based proactive caching to minimize the long-term energy used in wireless networks. The authors of [42] studied a two-level network hierarchical caching, to serve end-user file requests. A scalable deep reinforcement learning (DRL) approach based on hyper deep Q-networks (DQNs) was developed to provide an adaptive caching policy that leverages the interaction between caching decisions at different cache levels and reacts to the space-time evolution of file requests. They showed that their DRL algorithm can perform close to the optimal policy. The authors of [43] explored the usage of deep RL-based framework with Wolpertinger architecture for content caching at the base station for maximizing the long-term cache hit rate with no knowledge of the content popularity distribution. In [48], RL-based proactive caching was studied in the context of 5G networks. The authors focused on developing RL algorithms that learn the unknown popularity profiles, as well as the space-time popularity dynamics of user file requests. In particular, Joint consideration of global and local popularity demands along with cache-refreshing costs was impeded in the developed algorithm. Users' mobility adds to the complexity of the design of efficient proactive caching algorithms. To tackle such complexity, the authors of [45] considered proactive caching for vehicular networks. They developed a deep RL-based resource allocation policy that accounts for the vehicles' mobility and the hard service deadline constraint. The authors of [46] designed D2D caching strategies using multi-agent reinforcement learning. Specifically, multi-armed bandit problem and Q-learning are used to learn how to coordinate the caching decisions. Storing popular files reduces the overall throughput. However, the delay relation to caching is more complicated and not well studied. In particular, queue delay is more related to the scheduling interaction between cached and uncached files. Also, as videos can be segmented, sending later segments of videos can be less time-critical. As such, studying the effect of segmenting video caching decisions under the assumption that the back-haul is congested is crucial to improve the average transmission delay. Moreover, there is a need to study the effect of the segment's position in the file on the backhaul delay in comparison to segment's popularity.

2.4 Mobility in Proactive Caching

The authors of [7] studied a cache-aided small-cell system where users are moving through the network. They proposed an algorithm that minimizes the total average delay of all users and takes into account user mobility as well as geographical demand distribution. The authors of [49] proposed a novel scheme for UAV-enabled proactive caching at the users. Specifically, where a UAV is dispatched to serve a group of ground nodes and the UAV proactively transmits the files to a subset of selected ground nodes that serve the user requests directly. They showed that there exists a fundamental trade-off between the time required for the UAV to transmit the files to their designated caching ground nodes and the average time required for serving one file request.

2.5 Applications for Caching

The works in [22,50–53] provided more practical applications of coded caching. In [22], the authors studied the K-user broadcast channel (BC) and showed that caching combined with a rate-splitting broadcast approach can reduce the need for channel state information at the transmitter. In [51], the authors proposed a framework for edge-facilitated wireless distributed computing in which several MUs are connected to an access point. In [52], the authors studied the coded caching design for wireless networks with unequal link (the backhaul networks of LTE-A or 5G system) rates. In [50], the authors proposed a two-hop wireless network for video multicasting using coded caching. In [53], the authors generalized the cache-aided coded multicast problem to specifically account for the correlation among content files, such as the one between updated versions of dynamic data.

Chapter 3

Coded Caching for Time-Varying Files Popularities and Asynchronous Delivery

Abstract

[1]Data communication has seen exponential growth recently, and it currently dominates wireless communication. As a result, proactive caching was developed to minimize peak traffic rates by storing content, in advance, at different nodes in the network. We consider proactive caching for a broadcast wireless network with one central hub such as a satellite (ST) and K associated mobile units (MUs) such as mobile mini-ground stations or end users. The ST has a library of files, and the MUs demands are assumed to be limited to this library, while the popularity of the library files changes over time. We assume that the MUs demands arrive at different times, and hence, asynchronous file delivery is necessary. We propose a new scheme that minimizes the files delivery sum rate and show that we can use the file delivery messages to proactively and constantly update the MU finite caches. We show that this mechanism reduces the downloaded traffic of the network. The proposed scheme uses index coding to jointly encode the delivery of different demanded files with the cache updates to other MUs to follow the changes in the files popularities. An offline optimization of the delivery sum rate of the scheme is proposed, where it requires knowledge of the files popularities across the whole transmission period. In particular, the problem is formulated as a linear program and the optimal caching is obtained numerically. Moreover closed form solutions to two special cases are derived and a lowerbound to the achievable delivery sum rate is developed. Numerical results show the benefits of the proposed scheme over conventional caching schemes, in terms of reducing the delivery sum rate.

[1]This chapter was published as "Coded Caching for Time-Varying Files Popularities and Asynchronous Delivery", in *IEEE Open Journal of the Communications Society* .

3.1 Introduction

Cellular traffic has shifted over the past decade from mainly locally generated instantaneous traffic (voice calls) to centrally generated delay-tolerant bulks of traffic (data driven communication) [3]. This necessitated the development of new information theoretic analysis and new algorithms for communication networks. One key characteristic of data communication is that the delay between the time of data generation and the time of data demand at the receivers is mostly large. This enables the data to be stored midway in different nodes along the path from the transmitter to the receiver to relieve pressure of congested links. Future expansions of future generation cellular networks are targeting satellite integration with terrestrial networks to increase coverage and availability. Satellite links have broadcast nature and introduces further delays; thus, strengthening the case for midway storage (or caching). Moreover, the shift from voice to data traffic contradicts the classical assumption of network analysis that messages are generally independent. This assumption is not accurate for data communication, especially in broadcast networks, where a piece of data is generally demanded by multiple users over time. Proactive caching is an efficient technique to reduce the peak traffic rate and the delivery sum rate (the total transmission rate of the BS through out the predetermined delivery window). This is achieved by storing parts of the popular content, at various nodes in the network, ahead of their demand by different mobile units (MUs). Practically, proactive caching can minimize the total cost of transmission, as transmitters can optimize the time of performing proactive caching to be in less congested times (e.g. overnight) where the resources are in abundance.

Wireless networks with caches have been considered extensively in the recent literature [4, 7–11, 13–22, 25–27, 50]. The proactive caching research focuses on two main caching strategies. The first is storing raw files in respective caches [4, 7, 8]. The second is coding of different files together (e.g., index coding [1]) to optimize the usage of the resources [9–11, 13–22, 25–27, 50]. The authors of [4] proposed the idea of small-cell network cloud as a technique to increase cloud capacity, relieve the backhaul constraints[2] and increase the peak rate through content caching. Mobility and proactive caching are considered in [7], where a cache-aided small-cell system is considered while users are moving through the network and the content demand distribution is assumed to be known. The proactive content placement at different small-cells is optimized to minimize the total average delay of all users. In [8], the proactive content placement of a small cell is optimized while being oblivious of the demands statistics using reinforcement learning. In this chapter, we study index-coded caching and delivery for asynchronous delivery with changing file popularity and instantaneous file delivery. We consider the general case of change in file popularities. We study coded caching for a K-user network with local caches at the receivers. The transmission from a broadcast Satellite (ST) is assumed to occur over two phases, the placement and delivery phases. The users demands are assumed to be *asynchronous*, i.e., the delivery phase is composed of multiple time slots, where each mobile user (MU) demands a file delivery in a different time slot, and these demands are assumed to be unknown at the placement. The remainder of the chapter is organized as follows. In Section 3.2, we review the most related work

[2]Wireless backhaul are links that connects the base-stations to the core network. Their capacity are hard to increase because of cost and physical limitations.

to our contribution. In Section 5.2, the system model is described and the expected delivery sum rate is formulated. In Section 3.4, the proposed caching scheme is explained and the structure of the delivery plus update messages is provided. The resulting delivery rate is formulated in Section 6.5 as an optimization problem over the cached part sizes of all time-slots. The scheme size limitations are embedded in the optimization problem as linear constraints. Numerical results and performance measures for the scheme are provided in Section 3.9.

Notation: We use calligraphy fonts, i.e., $\mathcal{S}$, to denote sets. We use $A \setminus B$ to denote the information in A that is not available in B and $A \oplus B$ to denote the the bitwise XOR of A and B. We use $\binom{\mathcal{S}}{k}$ to denote the set of all subsets of $\mathcal{S}$ with size k, while we use ${}^{L}\mathcal{C}_k$ to denote the number of combinations when choosing k out of L. We use $\oplus$ to denote bit-wise XOR operation. We use $O(.)$ to denote the worst case complexity order.

3.2 Related Work and Motivation

Coded caching was first introduced in [9], where a two phase communication model that mirrors the probabilistic and broadcast message characteristics was developed. The authors studied a broadcast channel where transmission occurs over two phases. A number of independently generated messages, each corresponding to one piece of content, is available during the first transmission phase. During the second transmission phase, each receiver has a deterministic demand for one of the messages. The main difference compared to traditional network models is that the transmitter and the receiver need to optimize the channel use for the two phases

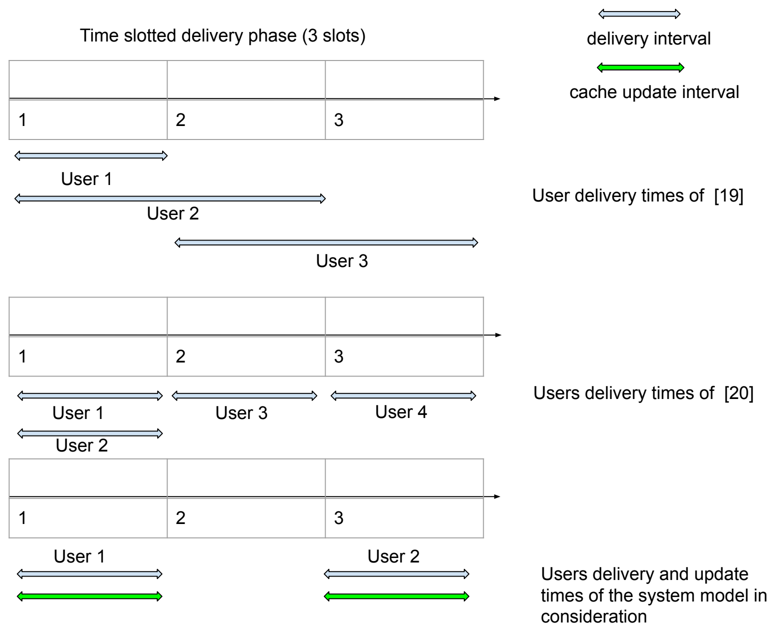

Time slotted delivery phase (3 slots)
delivery interval
cache update interval
1
2
3
User 1
User 2
User 3
User delivery times of [19]
1
2
3
User 1
User 3
User 4
User 2
Users delivery times of [20]
1
2
3
User 1
User 2
Users delivery and update times of the system model in consideration

Figure 3.1: Comparison of times and delivery deadlines of different Asynchronous Coded Caching work

jointly (the demand is probabilistic for the first phase) rather than for a single delivery phase. The authors presented a novel coded caching scheme that exploits the broadcast nature of the channel and index coding to reduce the transmission rate. They introduced an information-theoretic framework for the analysis of cache-aided communication in the context of broadcast channels and showed that the availability of caches in a broadcast setting provides a coded multi-casting gain. This work was extended to the case of non uniform demands in [10,11]. The authors of [10] proposed a near optimal coded caching scheme, whereas the authors of [11] derived a new information-theoretic lower bound on the expected transmission rate and a scheme that provides an expected transmission rate that is at most a constant factor away from the lower bound. On the other-hand, the authors of [14] focused on the case of uncoded placement and obtained the exact rate-cache size trade-off. Furthermore, some work developed decentralized schemes for the coded caching problem such as the works in [15, 16]. Coded caching was then extended to different wireless network models. Interference channel with coded caching was studied in [17–19] and coded caching within wireless device-to-device (D2D) network was studied in [20], [21]. Moreover, the work in [22, 50] provided more practical applications of coded caching.

The caching work can be categorized into two groups in terms of delivery (synchronous and asynchronous) and in terms of file popularities (changing and non changing). Synchronous delivery with no change in file popularities paired with the use of coded caching and/or delivery was studied in [9–11, 13–22, 50]. Asynchronous delivery with changing file popularities but with uncoded caching and uncoded delivery was studied in [4, 7, 8]. The use of index-coded caching and/or index-coded delivery for asynchronous delivery is not well explored in the literature. However,

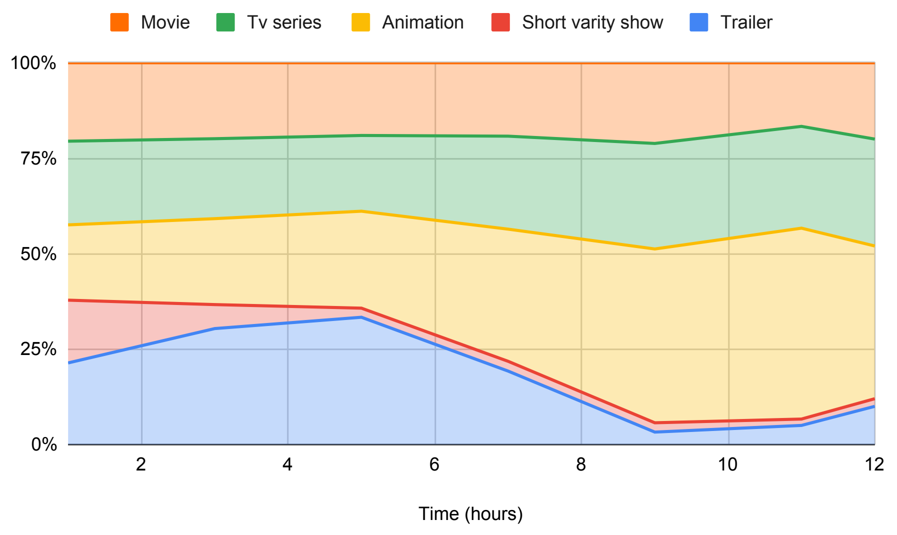


Figure 3.2: Comparison of relative popularity of different requested movie types across daytime.

asynchronous delivery is more dominant in practice, and a change in file popularities is expected over long delivery phases. Caching for asynchronous coded delivery was studied in [25], and [26] without considering the change in file popularities. The authors studied the asynchronous coded caching for network with constant file popularities and time window (deadline) for delivery. The authors of [26] studied the case where all receivers set an equal deadline for delivery, whereas the authors of [25] studied the case where the deadlines are different. On the other-hand, the authors of [27] studied the asynchronous delivery in the case a base station has to satisfy the user demands instantaneously.

The caching schemes of the previous two papers used the time overlap between users to send coded parts of the requested files (Fig. 3.1). However, since the cache hit rate for a large content library is generally low [54], the low cache hit rate renders the users requests that can be satisfied from the cache, in most applications in a wireless setting, distant in time. This is a huge challenge for the usage of synchronous coded caching. Moreover, in most applications users expect their requests to be satisfied in a short time window. In such case their coded messages can not satisfy more than one user simultaneously. Coded messages however can be used to update other user caches. This can be beneficial in the case that the content server learns the files popularities and users preferences over time or in the case that file popularities changes over time. Applications with changing file popularity is a case not previously considered in a coded caching setting despite its prevalence in practical scenarios. An example is traffic information which change fast. Another is a weather app that stores information about daily weather changes. Users often check the weather of the next hour or few hours; as such, content popularity changes over time. Video streaming is another example for strong changes in file popularities over short periods of time. The authors of [55] studied the demand patterns of video streaming and showed the change in patterns over time. Fig. 3.2 shows the relative popularities of different video genres throughout the day. Content popularity variations are stronger when localized, for example, downtown servers serve businesses at daytime and leisure seekers at night. Residential neighbourhoods' servers serve schools and local businesses in the morning and normal residents at night. Consequently, any overnight caching (placement phase) is not optimal for a whole day (delivery phase) and the cache needs to be updated throughout the delivery phase.

In [27], the authors provided a coded scheme for synchronous delivery and cache update. They studied a special case where popular files popularities is fading at constant rate.

Index coding [1] transmission consists of a set of W independent messages available at the transmitter

$$\mathcal{M} = \{M_1, M_2, ..., M_V\}, \tag{3.1}$$

to be transmitted to a set of K receiving nodes

$$\mathcal{U} = \{U_1, U_2, ..., U_K\}. \tag{3.2}$$

The kth destination node U_k is interested only in a set of messages $\mathcal{M}_k \subset \mathcal{M}$ and has another set of messages $\mathcal{A}_k \subset \mathcal{M}$ as side information (antidotes). A receiving node does not demand a message that is already available to it, i.e., $\mathcal{M}_k \cap \mathcal{A}_k = \phi$[3]. The server knows the side information at each receiver. The goal of an index code is to broadcast the minimum amount of information such that each user gets the message it is interested in.

An index coding scheme $D_n(\mathcal{S}, n, R)$ consists of a finite alphabet $\mathcal{S}$, chosen such that its cardinality, $|\mathcal{S}| > 1$, a coding function, f, and a decoding function $g_{k,i}$, for each demanded message M_i by each receiving node U_k. The coding function f maps the messages to the sequence of transmitted symbols

$$f(M_1, M_2, ..., M_K) = S^n, \tag{3.3}$$

where $S^n \in \mathcal{S}^n$ is the sequence of symbols transmitted over n channel uses. A message $M_k, k \in 1, 2, ...,$ V is a random variable uniformly distributed over the set $M_k \in \{1, 2, ..., |S|^{nR_v}\}$, where $R \in R_+^v$ [4] is the rate vector $R = (R_1, R_2, ..., R_V)$. The

[3] ϕ denotes an empty set

[4] R_+^S define a vector space, of size S, of positive real numbers.

decoding function at each receiving node is

$$g_{k,i}(S_n, A_k) = \hat{M}_{k,i}, \forall i, M_i \in \mathcal{M}_k. \tag{3.4}$$

It maps the transmitted symbols and the side information back to the needed messages. An achievable rate tuple $R = (R_1, R_2, ..., R_K) \in R_+^M$ exists if for each $\epsilon, \delta > 0$ there is a $(S, n, (\bar{R}_1, \bar{R}_2, ..., \bar{R}_K))$ coding scheme, for some S, n, such that $\forall w \in 1, 2, ..., V, \bar{R}_v \leq R_v - \delta$, and probability of error $P_e \leq \epsilon$. The probability of error is defined as $P_e = 1 - \text{Prob}[\hat{M}_{k,i} = M_i, \forall i, k$ such that $M_i \in \mathcal{M}_k]$.
The proposed system model is different from typical coded caching studied in the literature with one time slot delivery phase. Our model considers a more general setting when compared to [14] and [27]. We assume that the demands are non-uniform, with no constraints on the file popularities distributions, the MUs demands, the demands times, or the delivery times. We assume that the users are served by one ST or a number of STs that have the same file popularities.

The proposed scheme uses index coding to encode the missing information of a certain MU and update the caches of other MUs (joint delivery and update) even though the files are delivered at different time slots, and hence, reduces the expected delivery sum rate. The placement phase cache content and the content of the update messages are optimized according to the file popularities in order to minimize the expected delivery sum rate. The optimal solution for the delivery rate optimization of the scheme is obtained offline (at the placement phase) using linear programming, where the index coding problem is formulated as a linear program. Numerical results are provided to show the merits of the proposed scheme, over conventional caching schemes, in terms of reducing the delivery sum rate. The contributions of this chapter can be summarized as follows:

- We formally introduce and formulate the problem of asynchronous delivery for coded caching with changing file popularities and instantaneous delivery.
- We propose a scheme for asynchronous delivery and popularity-aware local caches update with minimal expected delivery rate compared to general schemes in literature.
- We develop a lowerbound to the achievable delivery sum rate.
- We derive a closed form solution to the case of two user network and the case of K user network with fading demand probability.

3.3 System Model

We consider a broadcast channel, where a Base Station (BS) communicates with K mobile units (MU). The BS and MUs are equipped with one antenna each and have limited size caches. The BS has a library of L cached N bit-files, $\mathcal{F} = \{F_1, F_2, ..., F_L\}$. The MUs may demand one of the library files at a time, where users demands are independent.

The probability that a file F_i is demanded at the jth time slot by the jth MU is denoted $P_{F_i,j}$, over the rest of the chapter we will use $P_{F_i,j}$ and $P_{i,j}$ interchangeably. Let d_j be the demand of the jth MU, $\mathbf{D}_K = \{d_1, d_2, ..., d_K\}$ be the full-size demand vector of all K users, and $\mathbf{D}_j$ be the vector of demands arriving at (and before) time slot $j \leq K$ (i.e. demands of j users). At the placement phase, the demand vector is

Symbol	Description
F_i	The ith file in the library stored at the ST.
$\mathcal{F}$	The library of L files stored at the ST.
$F_{i,m}$	The mth part of the ith file.
$\mathbf{d}_j$	The jth demand at the jth time slot.
$\mathbf{D}_j$	Vector of the last j demands.
$P_{i,t}$	The demand probability of file ith file at time slot t.
P_{D_K}	The aggregate demand probability of the demand vector $\mathbf{D}_K$.
S_i^{ϕ}	The normalized size of the parts of the ith file that are cached at the placement phase.
$S_i^{\mathbf{D}_l}$	The normalized size of the parts of the $i$thfile that are cached at each MU after D_l demands have been satisfied by the ST.
$S_{i,j}$	The normalized size of the parts of the ith file that are cached at each MU at time j.
$\mathcal{S}_{\mathcal{F}}$	The set of all file parts $S_i, S_i^{D_1}, ..., S_K^{D_K}$.
R_t	The expected delivery rate at time-slot t
R_d	The total expected delivery rate that is optimized offline.
M_{DU}	The delivery plus update message sent by the ST to the MUs.
M_D	The delivery part of the message M_{DU}.
M_U	The update part of the message M_{DU}.

Table 3.1: Summary of the symbols used through out the chapter

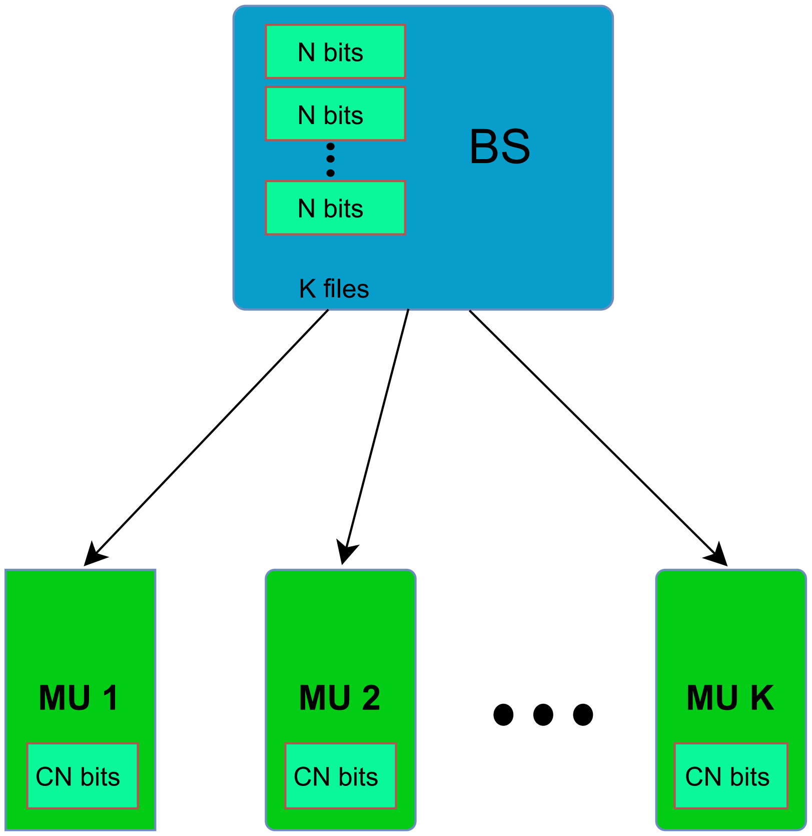


Figure 3.3: Network architecture.

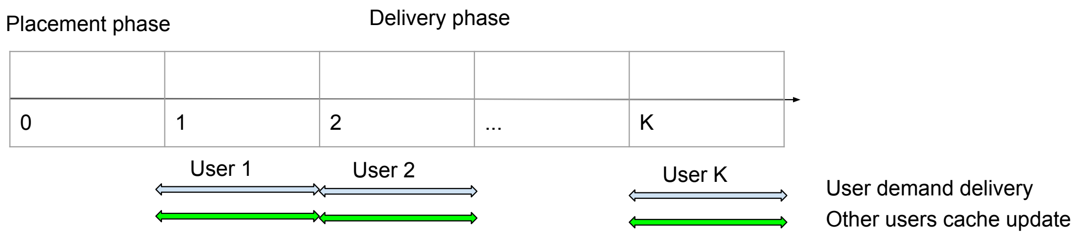


Figure 3.4: Placement and delivery phases

an empty vector denoted ϕ. Given that the demands are not known a priori, and each MU has only one demand $\{d_j \in \mathcal{F}, j \in 1, 2, ..., K\}$, there are L^K different possible full-size demand vectors over the entire delivery phase. Each possible demand vector has a probability $P_{D_K} = \prod_{j=1}^{K} P_{d_j,j}$. Each MU has a cache with size CN bits, i.e., can cache up to C files. A block diagram describing the system model is shown in Fig. 3.3.

The transmission from the BS to the MUs occurs over two phases, the placement phase and the delivery phase. During the first phase, i.e., the placement phase, the caches of the MUs are proactively filled with parts of the files of $\mathcal{F}$. During the second phase, i.e., the delivery phase, each MU demands one of the files F_i, $i \in \{1, 2, ..., L\}$ at time $t \in \{1, 2, ..., K\}$ and the BS sends the missing parts of this file that can not be extracted from the MU local cache. The MUs file demands are assumed to arrive at the beginning of different time slots (Fig. 3.4) within the delivery phase, and the BS satisfies each demand during its time slot. In each slot, one or more user demands are served while the other users' caches are updated. Each MU is assumed to have one demand within each placement-delivery window. The exact demands of the MUs are not known at the placement phase. Without loss of generality, we assume that the demand of MU j arrives at the jth timeslot. The demands occur according to known file popularities (demand probabilities) that are changing over time. The popularities are assumed to be known over all time slots.

3.3.1 Placement phase

Let S_i^{ϕ} be the normalized size (with respect to N) of the parts of file F_i that are cached at the placement phase (before the arrival of any demand) at each MU, i.e., $\sum_{i=1}^{L} S_i^{\phi} = C$. At the placement phase, each MU caches a part of each file i of size S_i^{ϕ}, $i = 1, 2, ..., L$, such that the MUs have different (possibly overlapping) but equally sized parts of each file, while the cached parts of different files are not necessarily equal in size. The reason behind storing different subfiles at the MUs in the placement phase is to help the MUs decode the update messages. If the missing part at the demanding user at any time slot is missing from the other users as well, the other users will not be able to decode the delivery plus update message.

3.3.2 Delivery phase

At the delivery phase, each user should receive the missing parts of their demanded files. Let $S_i^{\mathbf{D}_{j-1}}$ be the normalized size of the parts of file F_i that are cached at each MU at time j after satisfying the demand vector $\mathbf{D}_{j-1}$, where $j = 2, ..., K$. The expected delivery rate is minimized by optimizing the normalized sizes of the cached parts $S_i^{\phi}, S_i^{\mathbf{D}_j}$, $\forall\, i = 1, 2, ..., L, j = 1, 2, ..., K-1$. The delivery phase is assumed to occur over K time slots. The BS delivers the file demanded by the jth MU at the jth time slot. The delivery rate of this scheme depends on the cache content of the MUs and the MUs demands. Since the actual demands are not deterministic and not known a priori, we optimize the expected delivery rate. At the first time slot, if file i is demanded, the BS has to send the missing part $(1 - S_i^{\phi})$. Since the probability that file i is demanded at the first time slot is $P_{i,1}$, the expected delivery rate of the first time slot can be expressed as

$$R_1 = \sum_{i=1}^{L} P_{i,1}(1 - S_i^{\phi}). \tag{3.5}$$

In general, at time slot $j \geq 2$, the BS has to transmit $(1 - S_i^{\mathbf{D}_{j-1}})$ with probability given by $P_{i,j} \prod_{l=1}^{j-1} P_{d_l,l}$, where $d_l \in \mathcal{F}, \ \forall \ l = 1, 2, ..., j-1$, and $S_i^{\mathbf{D}_{j-1}}$ is the size of the cached part of file F_i at any MU after the demand vector $\mathbf{D}_{j-1}$. Accordingly, at any instant $j \geq 2$ the expected delivery rate can be expressed as

$$R_j = \sum_{d_1=1}^{L} ... \sum_{d_{j-1}=1}^{L} \Big(\prod_{l=1}^{j-1} P_{d_l,l} \Big) \Big(\sum_{i=1}^{L} (1 - S_i^{\mathbf{D}_{j-1}}) P_{i,j} \Big), \quad \forall \ j = 2, ..., K. \tag{3.6}$$

Our objective is to design the information transfer through the placement and delivery phases to minimize the expected delivery sum rate of the delivery phase

$$R_d = \sum_{j=1}^{K} R_j. \tag{3.7}$$

Throughout the rest of the chapter we will use MU and user interchangeably.

3.4 Time Varying Coded Caching Scheme

The main aim of the caching scheme is to minimize the expected delivery sum rate of user demands using proactive caching and proactive updates. The system has two main characteristics, asynchronous user demands and changing file popularities. These characteristics are a challenge for proactive caching and for index coding delivery. The caching scheme focuses on updating the user local caches to mirror popularity change. In other words, the caching scheme makes the more popular files more represented in the user caches. The main advantage of the scheme is that it performs the cache updates by using the delivery messages sent in the system without sending additional update message and without increasing the expected sum rate of the whole delivery phase. This is achieved using index coding. Moreover, given the changes in the file popularities, the design of the caching at the placement phase is optimized to these changes and do not depend only on the initial files popularities at the placement phase. The messages delivering missing parts of each file are used to update other MUs caches through joint index coding. The placement phase caching is optimized in conjunction with the delivery phase cache updates where the expected delivery rate of the delivery phase is the objective to be minimized.

3.4.1 Placement phase

The BS splits each file i into $K+1$ subfiles, $F_{i,m}, m = 1, 2, ..., K+1$. The first K subfiles are of a size S_i^{ϕ} each and are stored at the respective MUs, i.e., $F_{i,m}$ is stored at the mth MU. The last subfile is not cached ant any MU and is only available at the satellite. Its size is $\max(0, 1 - KS_i^{\phi})$. The last subfile size depends on S_i^{ϕ} which is chosen to minimize the expected delivery sum rate. The last subfile size is zero if S_i^{ϕ} is large ($KS_i^{\phi} > 1$). In this case, the file is completely cached/distributed among MUs' caches (Fig. 3.5). On the other hand, for small subfiles $KS_i^{\phi} \leq 1$, the file is not completely distributed among MUs' caches and the remaining part of the file $F_{i,K+1}$ of size $1 - KS_i^{\phi}$ is available at the ST.

3.4.2 Delivery and update phase

In the delivery phase, each user demands a file at a separate time slot. Responding to each demand, the BS sends a new delivery plus update message M_{DU}. The missing

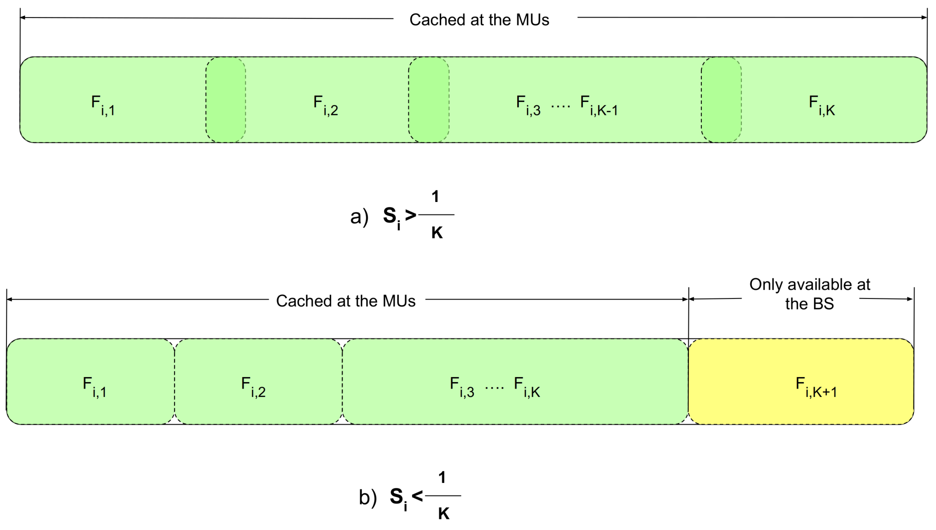


Figure 3.5: Cache placement comparison.

parts of the demanded file are encoded in the delivery part of the message M_D, the updates are encoded in the update part M_U, where M_{DU} is the XOR of M_D and M_U. As such, while delivering the missing part of the demanded file, the scheme uses the messages to update the caches of other users that still did not send their demands. In that sense, the scheme does not have an additional rate to that of delivery. For example, assume without the loss of generality, that the first MU demands the file F_i at the first time slot. The BS should deliver the missing subfiles of F_i that are not cached at the local cache of the first MU, i.e., send

$$\begin{aligned} M_D &= F_i \backslash F_{i,1} \qquad (3.8) \\ &= \{F_{i,2} \cup F_{i,3} ... \cup F_{i,G}\} \backslash F_{i,1}, \end{aligned}$$

$$G = \begin{cases} K, & KS_i^{\phi} \geq 1, \\ K+1, & KS_i^{\phi} < 1. \end{cases} \qquad (3.9)$$

Meanwhile, the BS needs to update the caches of the remaining $K-1$ MUs to reflect the new file popularities. The BS needs to increase the size of the part belonging to

Time	$P_{1,t}$	$P_{2,t}$	$P_{3,t}$
$t = 1$	0.5	0.25	0.25
$t = 2$	0.25	0.5	0.25
$t = 3$	0.25	0.5	0.25

Table 3.2: File demand probabilities over the delivery phase.

Time	S_1^{ϕ}	S_2^{ϕ}	S_3^{ϕ}
Placement (t=0)	0.5	0.25	0.25

Table 3.3: Cached parts sizes at the placement phase.

files that have seen a popularity increase by sending new parts to the MUs to cache. This has to be at the cost of flushing parts of the files that have seen a popularity decrease. Since, the update is XORed with delivery message sent to the first user, the update contents should be available at the first user for it to be able to decode the whole message and extract the missing subfile it demanded. The BS should send the following cache update

$$M_U = \{F_{1,1} \cup ... \cup F_{i-1,1} \cup F_{i+1,1} \cup ... \cup F_{L,1}\} \setminus F_{i,1}. \tag{3.10}$$

Accordingly, the BS transmits a delivery plus update message

$$M_{DU} = M_D \oplus M_U. \tag{3.11}$$

Time	$S_1^{D_1}$	$S_2^{D_1}$	$S_3^{D_1}$
$t = 1$	0.25	0.5	0.25

Table 3.4: Cached parts sizes after the first update (after the first time slot).

The first MU can decode the message M_{DU} and extract its demand $M_D = F_i \backslash F_{i,1}$ as it has the subfiles $M_U = \{F_{1,1}, F_{i-1,1}, ..., F_{i+1,1}, F_{L,1}\}$ in its caches. On the other hand, each of the other MUs can partially decode M_{DU} as each MU only has a part of M_D. Each MU can decode the part of the message that is XORed with a subfile it has in its cache and use the extract new subfile to update its cache (Fig. 3.6). The previous message (4.5) and subsequent cache update can be regarded as a cache swap between the first MU and the other MUs where some subfiles cached at the first MU are swapped with subfiles cached at other MUs.

Delivery plus Update message example: three users, three files:
Assume that we have a system that is composed of three users and three files. Assume that the file demand probabilities are as in Table 3.2.

Assume that the first user demands F_1 at the first time slot. The file parts are cached at the placement phase according to Table 3.3. The BS needs to send $F_1 \backslash F_{1,1}$ to the first user while sending an update to the second and the third user to change the cached sizes to be as in Table 3.4

The corresponding delivery plus update message is

$$M_{DU} = [F_{1,2} \cup F_{1,3}] \oplus [F_{2,1}, F_{3,1}]. \tag{3.12}$$

A depiction of the cache content and delivery plus update message is shown in Fig. 3.6.

The delivery rate of the delivery plus update message at the first time slot is equal to the size of the missing part of F_1 at the first user or 0.5. The expected delivery rate of the second and the time slot is

$$R_2 = P_{2,1}(1 - S_1^{D_1}) + P_{2,2}(1 - S_2^{D_1}) + P_{2,3}(1 - S_3^{D_1}) = 0.625. \tag{3.13}$$

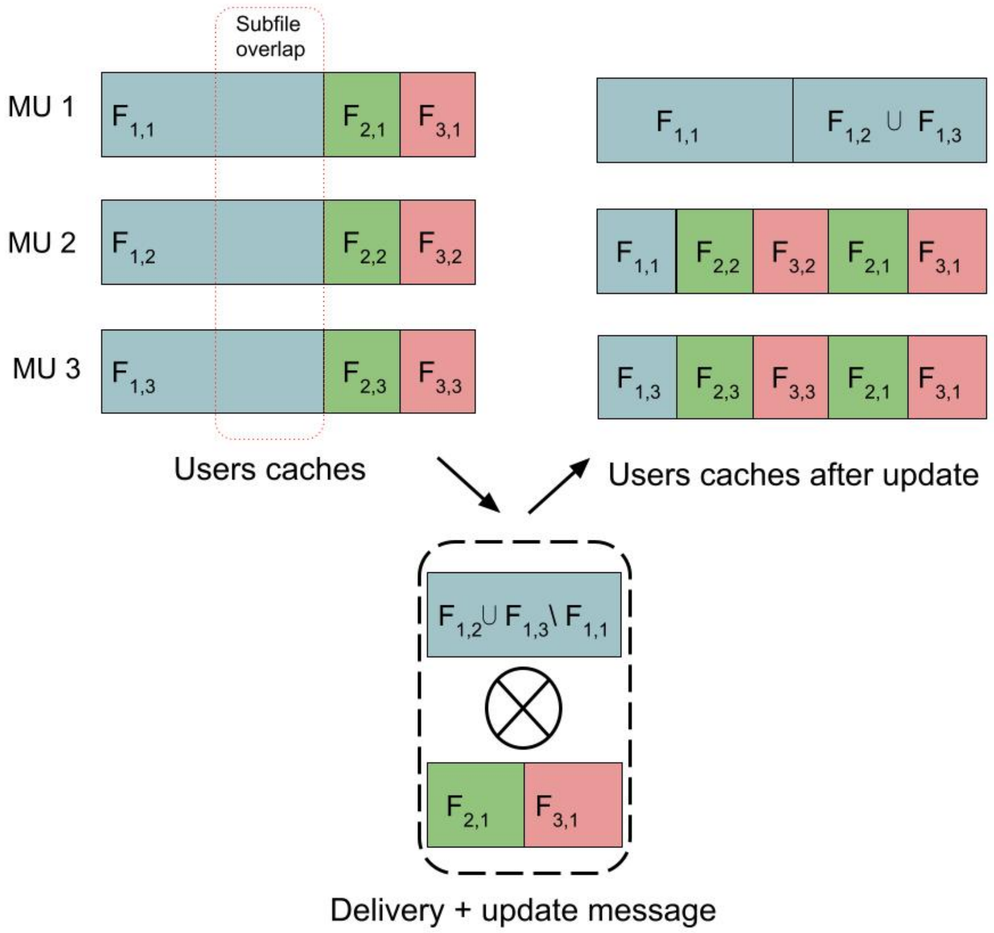

Subfile overlap
MU 1
MU 2
MU 3
$F_{1,1}$ $F_{2,1}$ $F_{3,1}$
$F_{1,2}$ $F_{2,2}$ $F_{3,2}$
$F_{1,3}$ $F_{2,3}$ $F_{3,3}$
Users caches
$F_{1,1}$ $F_{1,2} \cup F_{1,3}$
$F_{1,1}$ $F_{2,2}$ $F_{3,2}$ $F_{2,1}$ $F_{3,1}$
$F_{1,3}$ $F_{2,3}$ $F_{3,3}$ $F_{2,1}$ $F_{3,1}$
Users caches after update
$F_{1,2} \cup F_{1,3} \setminus F_{1,1}$
$F_{2,1}$ $F_{3,1}$
Delivery + update message

Figure 3.6: Example of delivery plus cache update message.

On the other hand, if no cache update is performed, the expected delivery rate of the second time slot would be

$$R_2 = P_{2,1}(1 - S_1^{\phi}) + P_{2,2}(1 - S_2^{\phi}) + P_{2,3}(1 - S_3^{\phi}) = 0.6875. \tag{3.14}$$

The reduction of the delivery rate comes with no increase in the delivery rate of the first time slot. The expected sum rate depends on the files parts sizes S_i^{ϕ}, $S_i^{\mathbf{D}_{j-1}}$, $j = 2, 3, ..., K,\ i = 1, 2, ..., L$. The parts sizes are chosen to minimize the expected sum rate. In the next section, we present an offline (at the placement phase) optimization of the scheme's expected delivery sum rate in detail and formulate the cache sizes and updates optimization. The expected delivery rate minimization is formulated as a linear program.

3.5 Offline Delivery Rate Optimization

The delivery and cache update described in the previous section enables the users caches to be partially updated according to the change in file popularities, which decreases the expected delivery rate. However, since the size of the update message M_U is limited by the size of the delivery message M_D, it is crucial to optimize the use of the update messages and the placement phase caching to minimize expected delivery rate of the scheme. These placement and update limitations are formulated as constraints over the cache sizes $S_i^{\phi}, S_i^{\mathbf{D}_{j-1}}, j = 2, 3, ..., K, i = 1, 2, ..., L$, of all time-slots of the placement and delivery phases. The offline optimization of the scheme expected delivery rate aims to minimize the expected delivery over all time slots jointly, under the assumption that the file popularities changes over the delivery phase are known at the placement phase. The expected delivery rate can be formulated as,

$$\begin{aligned} R_d =& \sum_{i=1}^{L} P_{i,1}(1 - S_i^{\phi}) + \sum_{d_1 \in \mathcal{F}} \sum_{i=1}^{L} P_{d_1,1} P_{i,2}(1 - S_i^{d_1}) + ... \\ & + \sum_{d_1 \in \mathcal{F}} ... \sum_{d_{K-1} \in \mathcal{F}} \sum_{i=1}^{L} P_{d_1,1} ... P_{d_{K-1},K-1} P_{i,K}(1 - S_i^{\mathbf{D}_{K-1}}). \end{aligned} \tag{3.15}$$

The expected delivery rate minimization (3.15) would be an unconstrained optimization problem if the caches can be completely changed at any instant.

However, since the scheme uses a limited size delivery plus update messages and the update is constrained by the current cache content at any instant, the expected delivery rate minimization is a constrained optimization problem where the decision variables S_i^{ϕ}, $S_i^{\mathbf{D}_{j-1}}$, $j = 2, 3, ..., K$, $i = 1, 2, ..., L$, are bounded. In the rest of the section, we will formulate the update constraints. First the formulation for a three MUs system is given as an example in subsection 3.5.1.1, then the formulation will be generalized to the K MUs case in subsection 3.5.1.2. Finally, we will formulate the expected delivery sum rate optimization problem in subsection 3.5.2.

3.5.1 Constraint formulation

In this section, we will present the cache size change limitations of the scheme. They are formulated as linear constraints on the expected delivery sum rate optimization problem.

3.5.1.1 Constraint formulation (3 MUs, 3 files example)

In this part, for the ease of notation, we give an example for a deterministic demand case. We will drop the demand vector notation D_j for the file parts sizes $S_i^{D_j}$ and use $S_{i,j}$ to represent the size of the part cached of file i at time slot j at any MU. To elaborate the update dynamics and size constraints, consider a system that consists of three MUs and three files $\{F_1, F_2, F_3\}$. The cached parts of F_i at the first, second and third MUs are $\{F_{i,1}, F_{i,2}, F_{i,3}\}$, respectively, and all have the same size $S_{i,j}$ at time slot j. Assume that F_1 is demanded by the second MU at the second time slot.

The size of the missing part of F_1 (i.e. $\{F_{i,2}, F_{i,3}\}$) to be delivered to the second user is $1 - S_{1,2}$ since the second MU has $S_{1,2}$ cached of the first file at the time of demand. Hence, the size of the delivery plus update message is $1 - S_{1,2}$, and the size of the update at the second time slot (difference between cached part size at the third and second time slots) of any file at any MU, is constrained as

$$S_{i,3} - S_{i,2} \leq 1 - S_{1,2}, \quad \forall i = 1, 2, 3. \tag{3.16}$$

Moreover, since the update is XORed with the delivery message $\{F_{i,2}, F_{i,3}\}$ and each MU has only a part of F_1 of size $S_{1,2}$, each MU can only decode part of the update message. The first MU can decode the part XORed with $F_{1,1}$, while the third MU can decode the part XORed with $F_{1,3}$, both are of size $S_{1,2}$. Therefore, the size update of any file at any MU is constrained as

$$S_{i,3} - S_{i,2} \leq S_{1,2}, \quad \forall i = 1, 2, 3. \tag{3.17}$$

Since the cache partition is symmetric. The cache updates for the first and the third files are constrained by the same constraints in (3.16) and (3.17). Finally, for the same arguments of (3.16) and (3.17), any combination of cache size changes has to be likewise constrained to the size of the decoded part at any MU $\min(S_{1,2}, 1 - S_{1,2})$. For example, considering the updates of the second and third files, their sum is constrained as,

$$S_{2,3} - S_{2,2} + S_{3,3} - S_{3,2} \leq S_{1,2}, \tag{3.18}$$

$$S_{2,3} - S_{2,2} + S_{3,3} - S_{3,2} \leq 1 - S_{1,2}. \tag{3.19}$$

Note that the constraints (3.17) and (3.16) are still necessary despite the existence of the constraints (3.18) and (3.19) because sizes of the cached parts are allowed to decrease, i.e. $S_{2,3} - S_{2,2}$ or $S_{2,3} - S_{3,2}$ can be of a negative value. Moreover, for the MU demanding file F_1 to be able to decode the update message, the XORed information with the delivered parts of F_1 should be locally available at its cache. In other words, the information sent to other users is a subset of the demanding user cache. For example, given the symmetry in the cache allocation, the information included in the update about files F_2 is constrained as

$$S_{i,3} - S_{i,2} \leq S_{i,2} \;\; \forall i = 1, 2, 3. \tag{3.20}$$

3.5.1.2 Constraint formulation (K MUs)

In the following, the previous constraints (3.16)-(3.21) will be generalized for the K MUs and L files case.

Generalizing (3.16) and (3.19), the cache update of each MU is limited to the size of the whole update message which can be formulated as

$$\begin{aligned} S_i^{\mathbf{D}_{j-1}} - S_i^{\mathbf{D}_{j-2}} &\leq 1 - S_{d_{j-1}}^{\mathbf{D}_{j-2}}, \\ &\forall i = 1, 2, ..., L, \; j = 2, 3, ..., K. \end{aligned} \tag{3.21}$$

The update message size constraint applies as well to any combination of cached file size

$$\begin{aligned} \sum_{F_i \in \mathcal{W}} S_i^{\mathbf{D}_{j-1}} - S_i^{\mathbf{D}_{j-2}} \;\; &\leq \;\; 1 - S_{d_{j-1}}^{\mathbf{D}_{j-2}}, \quad \forall i = 1, 2, ..., L, \\ & j = 2, ..., K, \; \mathcal{W} = \binom{\mathcal{F}}{l}, \; \forall l = 2, 3, ..., K. \end{aligned} \tag{3.22}$$

Generalizing to (3.17) and (3.18), the MUs' caches being updated has a part of the demanded file of size $S_{d_{j-1}}^{\mathbf{D}_{j-2}}$, and can only *decode* an equal sized part of the update message. In other words, the cache *update* of each MU is limited such that the *update* in file F_i at instant j (i.e., $S_i^{\mathbf{D}_{j-1}} - S_i^{\mathbf{D}_{j-2}}$) is smaller than or equal to $S_{d_{j-1}}^{\mathbf{D}_{j-2}}$ which is the size of the subfile it has in its cache. This can be represented as,

$$\begin{aligned} S_i^{\mathbf{D}_{j-1}} - S_i^{\mathbf{D}_{j-2}} &\leq S_{d_{j-1}}^{\mathbf{D}_{j-2}}, \\ &\forall\, i = 1, 2, ..., L,\ j = 2, 3, ..., K, \end{aligned} \tag{3.23}$$

where $\mathbf{D}_l = \{d_l, d_{l-1}, ..., d_1\}, d_l \in \mathcal{F}$. Similarly, the sum of any combination of the updates is smaller than the decoded part of the update message, which can be formulated as

$$\begin{aligned} &\sum_{F_i \in \mathcal{W}} S_i^{\mathbf{D}_{j-1}} - S_i^{\mathbf{D}_{j-2}} \leq S_{d_{j-1}}^{\mathbf{D}_{j-2}}, \forall i = 1, 2, ..., L, \\ &j = 1, 2, ..., K,\ \mathcal{W} = \binom{\mathcal{F}}{l},\ \forall l = 2, 3, ..., L. \end{aligned} \tag{3.24}$$

At any instant j, the MU demanding file l has no more than $S_i^{D_{j-1}}$ of file F_i. Therefore, the update cannot contain more than $S_i^{D_{j-1}}$ of $F_i, i = 1, 2, ..., L, i \neq l$ XORed with the missing part of F_l such that the MU demanding F_l can decode the update message and extract the missing part $F_l \backslash F_{l,1}$. As a result, and similar to (3.20), the update of any subfile at the other MUs' caches can be constrained as

$$\begin{aligned} S_i^{\mathbf{D}_{j-1}} - S_i^{\mathbf{D}_{j-2}} &\leq S_i^{\mathbf{D}_{j-2}}, \\ &\forall i = 1, 2, ..., K,\ j = 2, 3, ..., K. \end{aligned} \tag{3.25}$$

Finally, the cache size and file size limitations can be represented as

$$0 < S_i^{\phi} \leq 1, \quad \forall i = 1, 2, ..., L, \tag{3.26}$$

$$0 < S_i^{\mathbf{D}_{j-1}} \leq 1, \quad \forall j = 2, 3, , ..., K, \quad \forall i = 1, 2, ..., L, \tag{3.27}$$

$$\sum_{i=1}^{L} S_i^{\phi} = C, \tag{3.28}$$

$$\sum_{i=1}^{L} S_i^{\mathbf{D}_{j-1}} = C, \quad \forall j = 2, 3, , ..., K, \ \forall i = 1, 2, ..., L. \tag{3.29}$$

3.5.2 Delivery rate optimization problem formulation

The expected delivery rate (3.15) depends on all the demands probabilities and can be optimized to reflect all possible demand scenarios $\mathbf{D}_j, j = 1, 2, ..., K-1$, by optimizing the cached parts sizes and the updates sizes of all files over all time slots. The expected delivery rate minimization can be represented as follows

$$\begin{aligned} R_d = \quad & \min_{S_F} \ \sum_{i=1}^{L} P_{i,1}(1 - S_i) + \sum_{d_1 \in \mathcal{F}} \sum_{i=1}^{L} P_{d_1,1} P_{i,2}(1 - S_i^{d_1}) \\ & + \sum_{d_1 \in \mathcal{F}} ... \sum_{d_{j-1} \in \mathcal{F}} \sum_{i=1}^{L} P_{d_1,1} ... P_{d_{j-1},j-1} P_{i,j}(1 - S_i^{\mathbf{D}_{j-1}}), \\ & \text{Subject to } (3.23), (3.24), (3.21), (3.22), (3.25), (3.26), (3.27), (3.28), \\ & \quad \text{and } (3.29), \end{aligned} \tag{3.30}$$

where $\mathcal{S}_{\mathcal{F}}$ is the set of all file parts $S_i, S_i^{d_1}, ..., S_K^{d_1 d_2 ... d_K}$. The previous optimization problem is a linear programming optimization which can be solved by linear programming methods [56].

3.5.3 Complexity analysis

The dimension of the optimization problem of the provided scheme in (3.30) is K^K, and can be solved using interior point methods (IPM) [57] in $O((m+n)^{1.5}n^2L)$ processing time, where n is the number of variables and L is at most the number of bits used by the solver to represent the input. The quantity L is often defined as $L = \log(m) + \log(1 + d_{\max}) + \log(C_0)$ where $m = K^K(2^{K+1} - 1) + K^{K-1})$ is the number of constraint in the minimization problem, $d_{\max} = O(K^K)$ is the largest absolute value of the determinant of a square sub-matrix of the constraint coefficient matrix, and C_0 is a constant.

3.6 Lowerbound to The Delivery Sum Rate

In this section we provide a lowerbound on the expected delivery sum rate of the system model of consideration. Consider a network where a genie would provide the MUs with the required information to decode the entire message sent in the network and update their caches. The message size is constrained to the size of the missing information of the demanded file. For example if the message is

$$F_{i,x}, F_{i,y} \oplus F_{i,u}, F_{i,v}, \tag{3.31}$$

and an MU has only $F_{i,x}$, the genie will give the MU $F_{i,y}$ to obtain $F_{i,u}, F_{i,v}$. This is also equivalent to sending an independent update message to each MU at each time slot which the same size of the delivery message. The expected delivery sum rate of this network is a lowerbound for the expected delivery sum rate of any scheme for the network in consideration.

The cache size update of the genie aided update message size is constrained as

$$\begin{aligned}\sum_{F_i \in W} S_{i,j} - S_{i,j-1} \;\; &\leq \;\; \sum_{n=1}^{L} P_{n,j-1}(1 - S_{n,j-1}), \\ &\forall j = 2, ..., K,\ \mathcal{W} = \binom{\mathcal{F}}{l},\ \forall l = 2, 3, ..., K. \end{aligned} \tag{3.32}$$

The expected delivery sum rate of the genie aided network can be represented as

$$\begin{aligned} R_{genie} = \quad & \min_{S_F} \quad \sum_{j=1}^{K} \sum_{i=1}^{L} P_{i,1}(1 - S_{i,j}) \\ & \text{Subject to} \sum_{i=1}^{L} S_{i,j} = C,\ \forall j = 1, 2, ..., K \\ & \text{and (3.32).} \end{aligned} \tag{3.33}$$

The previous optimization problem is a mutli-period optimization and can be solved using interior point methods [57].

3.7 Closed Form Solutions

In this section, we present closed form solutions for the expected delivery sum rate optimization ,(3.31), for two special cases, posed in two theorems.

Theorem 1. *For a two MUs network, the following expected delivery sum rate is achievable*

$$R_d = \min \left\{ P_{2,1} + P_{2,2}, P_{1,2} + \min \left\{ \frac{1}{2}, 1 - 2P_{1,1}P_{1,2} \right\} \right\}. \tag{3.34}$$

Proof. See Appendix 3.10 □

Theorem 2. *For a K MU network with decreasing file popularity of the demanded files and equal file popularities of undemanded files, the following expected delivery sum rate is achievable*

$$R_{\mathrm{d}}(K) = \left(K - \left(\psi(K+1) + \gamma\right)\right) N, \tag{3.35}$$

where γ is the Euler-Mascheroni constant and ψ is the is the digamma function.

Proof. See Appendix 3.10 □

3.8 Mixed delivery

The previous sections showed how the proposed scheme serves asynchronous demands. The scheme can be modified to serve mixed mode demands where some of the demands are served in the same time slots and some are served in separate time slots. In this section, we will show the extension for a three users, three files network, and the asynchronous period precedes the synchronous period.

3.8.1 Placement phase

Assume without lose of generality that file one is demanded alone and files two and three are demanded together. The caching at the placement phase is performed in the same manner as the asynchronous case where the expected delivery rate is minimized for two delivery time slots instead of three

$$\{F_{1,11}, F_{1,12}, F_{2,11}, F_{2,12}, F_{3,11}, F_{3,12}\}, \tag{3.36}$$

$$\{F_{1,21}, F_{1,22}, F_{2,21}, F_{2,22}, F_{3,21}, F_{3,22}\}, \tag{3.37}$$

$$\{F_{1,31}, F_{1,32}, F_{2,31}, F_{2,32}, F_{3,31}, F_{3,32}\}. \tag{3.38}$$

3.8.2 Delivery and update

In order to satisfy the demand of the first file at the first time slot, the BS sends a delivery and update messages

$$M_D = \{F_{1,21}, O_{d1}, F_{1,31}, O_{d2}, F_{1,22}, O_{d3}, F_{1,32}, O_{d4}\}, \tag{3.39}$$

$$M_U = \{F_{2,11}, O_{u1}, F_{2,12}, O_{u2}, F_{3,11}, O_{u3}, F_{3,12}, O_{u4}\}. \tag{3.40}$$

where $O_{di}, i = 1, ..., 4$, $O_{ui}, i = 1, ..., 4$ are padding subfiles to accommodate for sub-file size difference.

After updating the MUs caches, the cache contents of the last two users become

$$\{F_{2,11}, F_{2,21}, F_{2,22}, F_{3,11}, F_{3,21}, F_{3,22}\}, \tag{3.41}$$

$$\{F_{2,12}, F_{2,31}, F_{2,32}, F_{3,12}, F_{3,31}, F_{3,32}\}. \tag{3.42}$$

Alternatively, for the ease of notation the caches can be represented as

$$\{F_{2,2}, F_{3,2}\}, \tag{3.43}$$

$$\{F_{2,3}, F_{3,3}\}. \tag{3.44}$$

Finally, at the second time slot the BS sends

$$\{F_{2,2} \oplus F_{3,3}\}, \tag{3.45}$$

to deliver the missing information of the second and the third file at the last two users. The update constraints for this case are listed below $\forall d_1 \in \{F_1, F_2, F_3\}$ and $i \in$

$\{1, 2, 3\}$:

$$S_i^{d_1} \leq 1 - S_{d_1}^{\phi}, \tag{3.46}$$

$$S_i^{d_1} \leq S_{d_1}^{\phi}, \tag{3.47}$$

$$\sum_{i, F_i \neq d_1} S_i^{d_1} \leq 1 - S_{d_1}^{\phi}, \tag{3.48}$$

$$\sum_{i, F_i \neq d_1} S_i^{d_1} \leq S_{d_1}^{\phi}, \tag{3.49}$$

$$S_i^{d_1} - S_i^{\phi} < S_i^{\phi}. \tag{3.50}$$

The expected delivery sum rate of the previous scheme can be represented as

$$\begin{aligned} R_d = \quad & \min_{\mathcal{S}} \quad \sum_i P_{i,1}(1 - S_i^{\phi}) \\ & \qquad + \sum_{d_j} P_{d_j,2} \max_{l=1,2}(P_{<d_j>+l,2}(1 - S_{<d_j>+l}^{d_j})) \\ & \text{subject to} \quad (3.26), (3.27), (3.28), (3.29), (3.46), (3.47), (3.48), \\ & \qquad (3.49), \text{and } (3.50). \end{aligned} \tag{3.51}$$

where and $< d_j >$ to denote the index i of the file F_i of the demand d_j. The minimization problem in (3.51) can be reformulated as

$$\begin{aligned} R_d = \ & \min \ \sum P_{i,1}(1 - S_i^{\phi}) + \sum_{d_i} P_{d_i,2} Z_{d_i} \\ & \text{subject to} \, Z_{d_i} \geq P_{<d_i>+1,2}(1 - S_{<d_i>+1}^{d_i}) \\ & \qquad Z_{d_i} \geq P_{<d_i>+2,2}(1 - S_{<d_i>+2}^{d_i}) \\ & \qquad (3.26), (3.27), (3.28), (3.29), (3.46), (3.47), (3.48), \\ & \qquad (3.49), \text{and } (3.50). \end{aligned} \tag{3.52}$$

where Z_{d_i} is an auxiliary variable. Similar to the optimization in (3.31), the previous optimization is a linear program and can be solved using linear programming methods [56].

3.9 Results

In this section, we provide results via system simulations to show the merits of the proposed scheme in terms of the expected delivery sum rate. Additionally, we compare the performance of the proposed scheme to the performance of two other schemes. The first is the uncoded offline caching scheme, where this caching scheme typically stores the most popular files at the placement phase [9]. The second is the caching last used file scheme (LSC) [58]; this is an online caching scheme where users store the most recent files sent to others users in the network. The LSC is beneficial in the case that a small group files have the highest popularities over a long period of time, but not in the case that the files popularities are rapidly changing. The efficiency of the proposed caching algorithms does not only depend on the change of the file popularities but also depends on how popularities or the demand distribution changes from time to time. The simulations are carried out for two scenarios that represent different behaviors of the change in popularities. The first is when the popularities are randomly changing according to

$$P_{i,t} = \frac{\theta_{F_{i,t}}}{\sum_i \theta_{F_{i,t}}} \;\; \forall t = 1, 2, ..., K, \; \forall i = 1, 2, ..., L, \tag{3.53}$$

where $\theta_{F_{i,t}} \sim U(0, 1)$. Figure 3.7 plots the probabilities change of five files over time according to distribution (3.53), and Fig. 3.8 shows the simulated probability distributions for three of the five files. The second scenario is when the files popularities are following a trend (i.e., some files have fading popularities and

Time	$P_{1,t}$	$P_{2,t}$	$P_{3,t}$	$P_{4,t}$
$t = 1$	0.2469	0.0056	0.4443	0.3032
$t = 2$	0.2697	0.0505	0.3707	0.3091
$t = 3$	0.3297	0.2667	0.2212	0.1824
$t = 4$	0.2878	0.3564	0.0944	0.2613

Table 3.5: File demand probabilities over the delivery phase.

Time	d_1	d_2	d_3	d_4
$t = 1$	2	2	4	3

Table 3.6: Worst case demand vector.

other have growing popularities). The demand probabilities in the second case are generated according to

$$P_{i,t} = \frac{i^t \theta_{F_{i,t}}}{\sum_{i=1}^{K} i^t} \;\; \forall t = 1, 2, ..., K, \; \forall i = 1, 2, ..., L, \tag{3.54}$$

where $\frac{i^t}{\sum_{i=1}^{K} i^t}$ represent the exponential growth/decline trend line. For example, for the fourth file ($i = 4$), the value $\frac{4^t}{\sum_{i=1}^{4} 4^t}$ grows with time while for the first file ($i = 1$) $\frac{1^t}{\sum_{i=1}^{4} 1^t}$ declines with time. $\theta_{F_{i,t}}$ is a smaller (on average) random change around the dominant trend line.

Fig. 3.7 plots the probabilities change over time of five files according to distribution (3.54), and Fig. 3.10 shows the simulated probability distributions

$S_3^{D_1}$	$S_2^{D_1}$	$S_3^{D_2}$	$S_4^{D_2}$
0.0326	0.0652	0.0543	0.5543

Table 3.7: Cached parts sizes of the worst case demand vector.

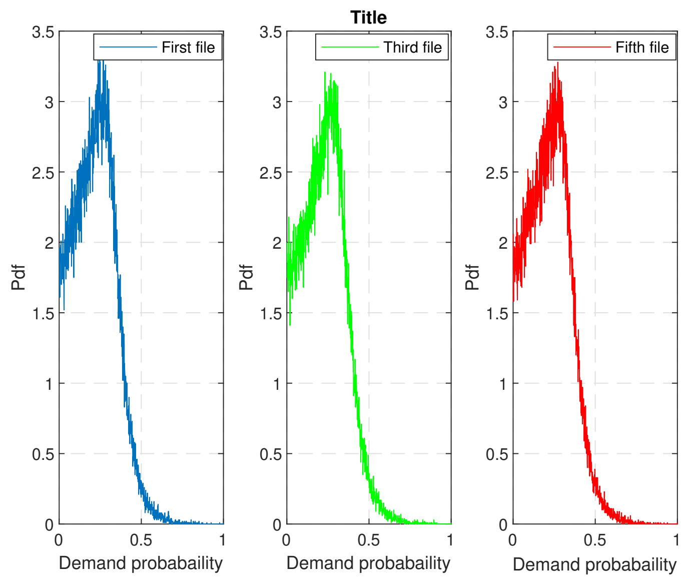


Figure 3.7: Simulated distributions of demand probabilities of the first scenario

for three of the five files. The simulation of the delivery rate was performed using Matlab, where the number of mont-carlo simulations was one million runs for each simulation. User demands were generated according to the above mentioned distributions for each simulation setting. Figure 3.9 and Fig. 3.12 show the expected delivery sum rate of the uncoded caching, LSC, offline coded caching schemes, and the genie aided lowerbound for the first and the second demand scenarios, respectively. The network is simulated for a cache size of one file at each user in both figures. The figures show the caching gain (reduction in delivery phase transmission rate) of the proposed coded caching in comparison to uncoded proactive caching and LSC. It can be observed that the caching scheme provides a significant reduction in delivery rate.

For example, for a five users, five files network in the second scenario (Fig. 3.12), the expected delivery sum rate if no caching is used is five for the case that the

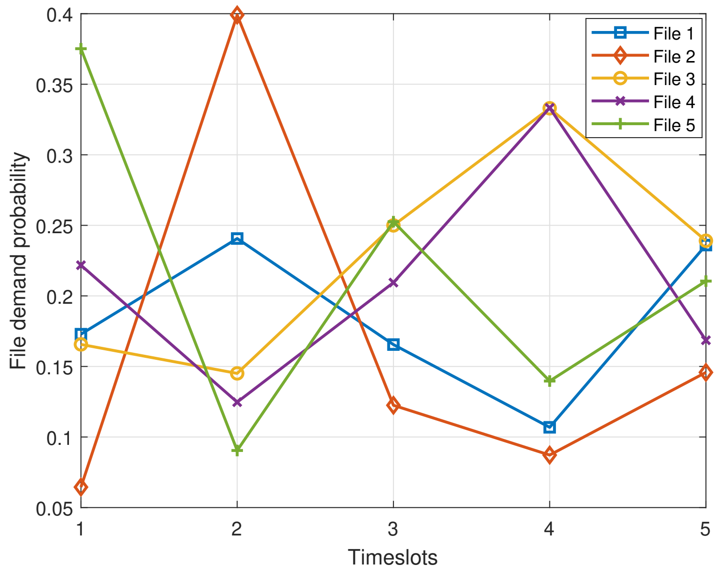


Figure 3.8: An instance of the first scenario demand probability trace for a five files system.

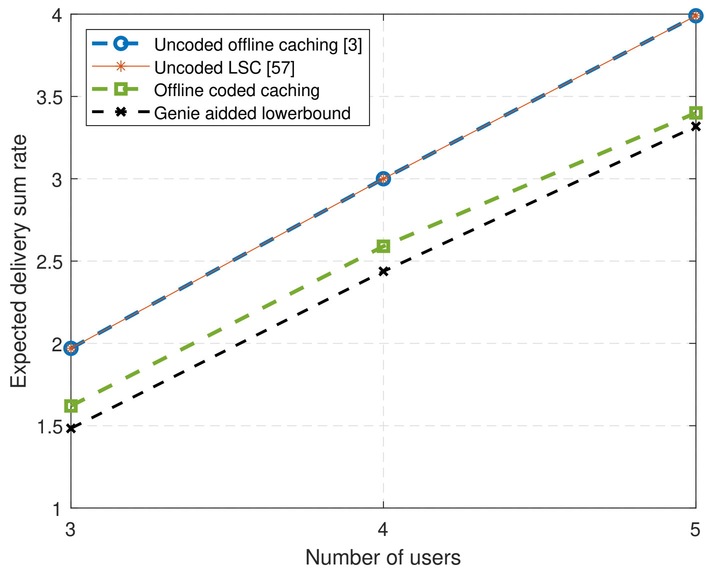


Figure 3.9: The performance of caching schemes in terms of the delivery rate for different number of users for the first demand scenario.

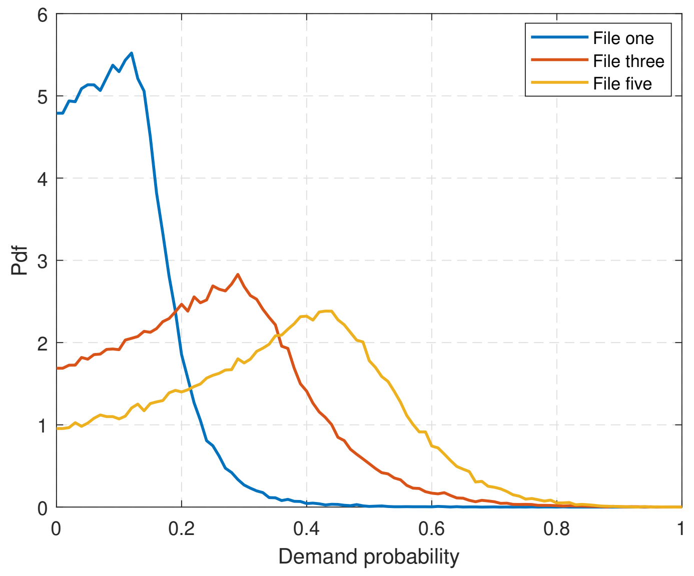


Figure 3.10: Simulated distributions of demand probabilities of the second scenario.

files popularity have the same distribution. The delivery rate reduction for using the proposed scheme is 50%, 37.5%, and 27.5% relative to no caching, uncoded caching, and LSC, respectively. The figures show that the proposed scheme achieves an expected delivery sum rate close to the lowerbound. Since that the presented lowerbound is aided with a large amount of side information, it is expected that the optimal delivery sum rate is closer to the proposed scheme than the presented lowerbound.

Moreover, both figures show a growing gain with the number of MUs in the system. The aggregate cache of the MUs grows with the number of MUs which gives more room for offloading more files to the MU side. It is observed that the performance of the proposed coded caching scheme is better in the second scenario compared to the first. This is due to the restrictions on the update message size which limits the

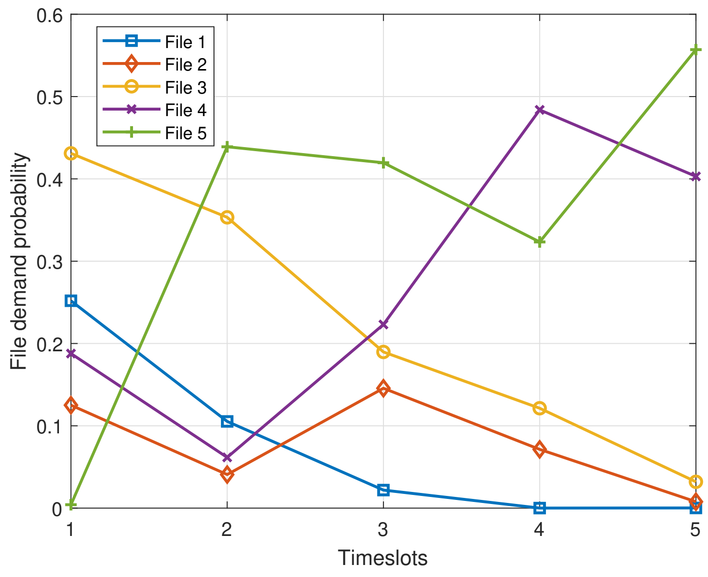


Figure 3.11: An instance of the second scenario demand probability trace for a five files system.

capacity of performing a very large update to the MUs caches compositions to track big and swift changes in the file popularities.

It is worth noting that in a practical setting the popularities changes would be more related to the trend line case (second scenario). Figures 3.13 and 3.14 show the change in the expected delivery sum rate with increasing the cache size for a four MUs network for the first and the second demand scenarios, respectively. The uncoded caching saves on average an amount equal to the cache size. The simulation shows that the proposed coded caching scheme is more beneficial when the cache size is small due to optimized usage of the limited cache size. The previous figures show the average performance of the proposed scheme. Meanwhile, the scheme has good performance in the worst case where the users demand the least popular files since

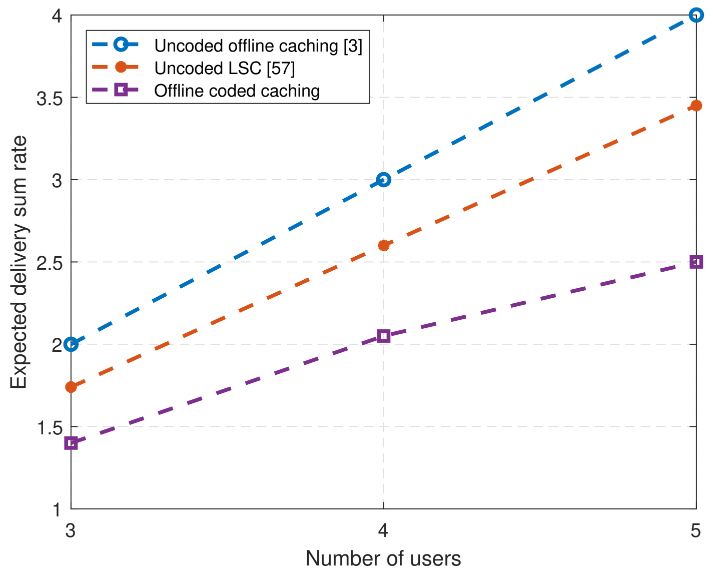


Figure 3.12: The performance of caching schemes in terms of the delivery rate for different number of users in the second demand scenario.

the scheme caches parts of each file. An example with file popularities according to Table 3.5, and worst case demand vector as in Table 3.6 is provided. The scheme caching decision is as described in Table 3.7 and the resulting worst case rate is 3.29 compared to 4 if the most popular caching is used and 3 if LSC is used. Fig. 3.15, on the other hand, shows the performance of the scheme for the special case of fading file popularities discussed above.

3.10 Conclusion

We studied a K MUs satellite broadcast network where the satellite has a pool of cached files, and the MUs demands are expected to be limited to that pool. The popularities of the files are changing over time. The proposed scheme jointly encodes the delivery of different demanded files with the cache updates to the MUs

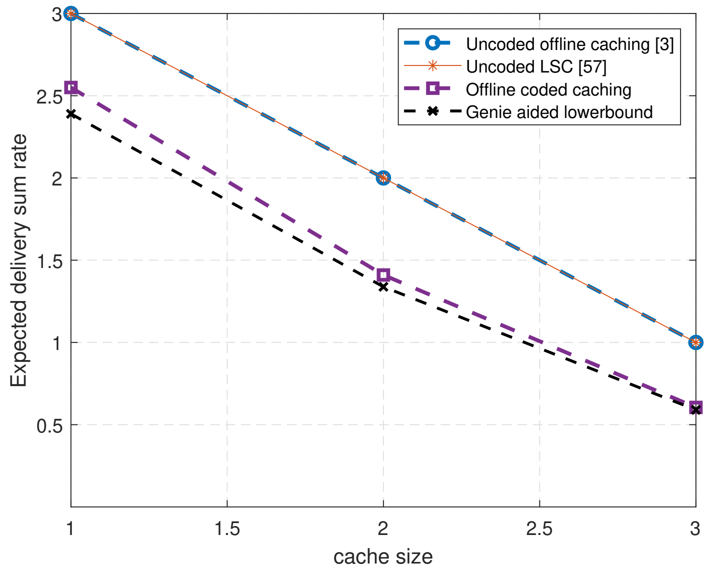


Figure 3.13: The performance of caching schemes in terms of the delivery rate with the change in cache size for the first demand scenario.

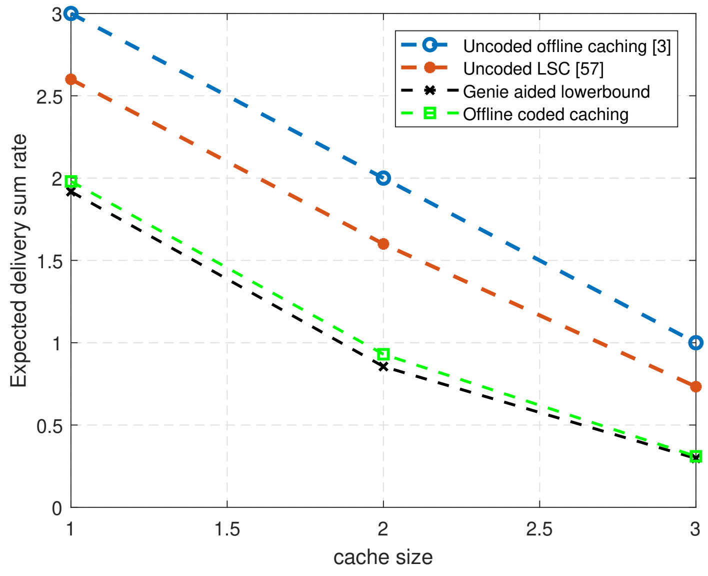


Figure 3.14: The performance of caching schemes compared to other caching schemes for different cache sizes.

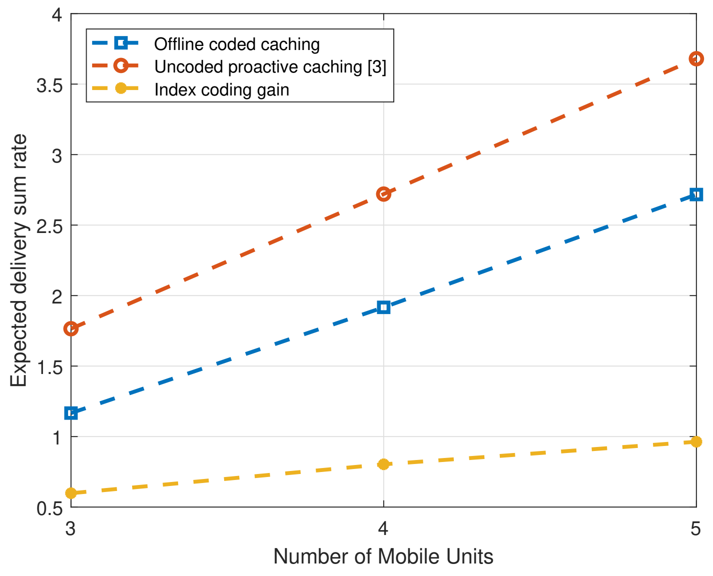


Figure 3.15: The performance of caching scheme for a special case of fading popularities compared to uncoded caching.

over different time slots to minimize the delivery sum rate.An offline optimization of the scheme is proposed. Our simulation results show that the proposed scheme significantly reduce the delivery rate for all cache sizes and different number of users in the system for different distributions/behavior of the files popularities. Specifically, the proposed scheme has better behavior when the files popularities follow a specific trend, which would be a typical scenario in a practical setting, and/or when the cache sizes are large relative to the number of files. We showed that proactively and constantly updating the MU finite caches reduces the downloaded traffic of the network.

Appendix 3.A

Theorem 3. *For a two MUs network, the following expected delivery sum rate is achievable*

$$R_d = \min\left\{P_{2,1} + P_{2,2}, P_{1,2} + \min\left\{\frac{1}{2}, 1 - 2P_{1,1}P_{1,2}\right\}\right\}. \tag{3.55}$$

Proof.

Placement Phase

In this phase, each user caches a part of each file of size S_1^{ϕ} and S_2^{ϕ}). The cache contents at the MU one and two are

$$\{F_{1,1}, F_{2,1}\}, \tag{3.56}$$

$$\{F_{1,2}, F_{2,2}\}, \tag{3.57}$$

respectively.

Delivery Phase

For a two files system we have two scenarios. The first is that one file is more popular for the entire delivery phase. In this case, it is cached at both users and no update is needed. Assume without loss of generality that file F_1 is more popular, then the delivery rate is

$$R_d = P_{2,1} + P_{2,2}. \tag{3.58}$$

The second case is that the most popular file changes through the delivery phase. If file F_1 at the first delivery time slot is requested, then the BS transmits

$$M_{DU} = F_1 - F_{1,1} \oplus F_{2,1}. \tag{3.59}$$

The second MU extracts $F_1 - F_{1,1}$, and then updates its cache by replacing a part of F_1 with $F_{2,1}$ as the popularity of file F_2 increases in the second time slot. As a result, the updated cache of the second MU has a part F_2 of size $2S_2^{\phi}$ and a part of F_1 of size $(1 - 2S_2^{\phi})$. At the second time slot, the BS transmits a part of size $2S_2^{\phi}$ if file F_1 is demanded by the second MU or a part of size $(1 - 2S_2^{\phi})$ if file F_2 is requested.

On the other hand, if file F_2 is demanded by the first MU at the first delivery time slot, then the BS sends

$$C = \{F_{2,2} \oplus F_{2,1}, F_{2,3}\}. \tag{3.60}$$

As such, the first MU extracts $F_2 \backslash F_{2,1}$, while the second MU updates its cache to include the whole file F_2 that becomes more popular for the coming second delivery time slot and caching it is instantly optimal. At the second time slot, the BS transmits nothing if file F_2 is requested by the second MU, while it transmits the whole file F_1 if it was requested by the second MU. The delivery phase sum rate for this scheme is

$$\begin{aligned} R_d &= P_{1,1}S_2^{\phi} + P_{2,1}S_1^{\phi} + P_{2,1}P_{1,2} + 2(P_{1,1}P_{1,2})S_2^{\phi} \\ &\quad + P_{1,1}P_{2,2}(1 - 2S_2^{\phi}). \end{aligned} \tag{3.61}$$

Given that $P_{1_j} = 1 - P_{2_j}$ and $S_1^{\phi} = 1 - S_2^{\phi}$, then

$$R_d = 1 + P_{1,2} - 2P_{1,1}P_{1,2} + (4P_{1,1}P_{1,2} - 1)S_2^{\phi}. \tag{3.62}$$

Hence, the minimum expected delivery sum rate can be formulated as

$$\begin{aligned} R_d &= \min_{S_2^\phi} 1 + P_{1,2} - 2P_{1,1}P_{1,2} + (4P_{1,1}P_{1,2} - 1)S_2^\phi \\ & \quad \text{subject to} \quad S_2^\phi \leq \frac{1}{2}. \end{aligned} \tag{3.63}$$

The solution of the expected delivery sum rate minimization problem in (3.63) depends on the sign of $(4P_{1,1}P_{1,2} - 1)$, as the problem represents a one-dimensional linear optimization problem. If $4P_{1,1}P_{1,2} \geq 1$, then the term $(4P_{1,1}P_{1,2} - 1)S_2^\phi$ is positive and the minimizing value of S_2^ϕ is 0, i.e., $S_2^{\phi^*} = 0$ and the resulting minimum delivery sum rate is given as

$$R_d = 1 + P_{1,2} - 2P_{1,1}P_{1,2}, \quad \text{if} \quad 4P_{1,1}P_{1,2} \geq 1. \tag{3.64}$$

On the other hand, if $4P_{1,1}P_{1,2} < 1$, then term $(4P_{1,1}P_{1,2} - 1)S_2^\phi$ is negative and the minimizing value of S_2^ϕ is $\frac{1}{2}$, i.e., $S_2^{\phi^*} = \frac{1}{2}$, and the resulting minimum delivery sum rate is given as

$$R_d = \frac{1}{2} + P_{1,2}, \quad \text{if} \quad 4P_{1,1}P_{1,2} < 1. \tag{3.65}$$

Similarly if $S_1^\phi \leq S_2^\phi$ at the placement phase, i.e., file F_2 is fully cached while file F_1 is partially cached at the combined cache of the two MUs, the delivery phase sum rate would be

$$R = P_{1,1}S_2^\phi + P_{2,1}S_1^\phi + P_{1,1}P_{1,2} + P_{2,1}P_{1,2}, \tag{3.66}$$

and given that $P_{2,t} = 1 - P_{2,t}$ and $S_1^\phi = 1 - S_2^\phi$, then

$$R_d = (2P_{1,1} - 1)S_2^\phi - P_{1,1} + P_{1,2} + 1. \tag{3.67}$$

Hence, the minimum expected delivery sum rate if parts of files F_1 and F_2 are cached at the placement phase and $S_1^\phi \leq S_2^\phi$ can be formulated as

$$\begin{aligned} R_d &= \min_{S_2^\phi} \quad (2P_{1,1}-1)S_2^\phi - P_{1,1} + P_{1,2} + 1 \\ & \quad \text{subject to} \quad S_2^\phi \geq \frac{1}{2}. \end{aligned} \tag{3.68}$$

Since file F_1 is more popular at the first delivery time slot, $P_{1,1} > \frac{1}{2}$ and $(2P_{1,1}-1)$ is positive. The minimizing value for S_2^ϕ is $\frac{1}{2}$, i.e. $S_2^{\phi^*} = \frac{1}{2}$. Accordingly, the minimum delivery sum rate is

$$R_d = \frac{1}{2} + P_{1,2}. \tag{3.69}$$

Finally, the minimum delivery sum rate can be expressed as

$$R_d = \min\left\{P_{2,1} + P_{2,2}, P_{1,2} + \min\left\{\frac{1}{2}, 1 - 2P_{1,1}P_{1,2}\right\}\right\}. \tag{3.70}$$

□

Appendix 3.B

Theorem 4. *For a K MU network with decreasing file popularity of the demanded files and equal file popularities of undemanded files, the following expected delivery sum rate is achievable*

$$R_\mathrm{d}(K) = (K - (\psi(K+1) + \gamma))\, N, \tag{3.71}$$

where γ is the Euler-Mascheroni constant and ψ is the is the digamma function.

Proof. In this section, we derive the closed form of the expected delivery sum rate for the case when the files popularities are equal for all undemanded $(K-t)$ files at each time slot t.

The Placement Phase

The BS splits the ith file into a number of $K(K-1)$ subfiles $\{F_{i,lh}; i=1,2,...,K; l=1,2,...,K; h=1,2,...,K-1\}$. In the placement phase, MU m stores subfiles $\{F_{i,mh}; i=1,2,...,K; h=1,2,...,K-1\}$.

The Delivery Phase

Assume without loss of generality, that the first MU requests file F_1 at the first time slot. The BS transmits

$$\{F_{1,21}, F_{1,22}, ..., F_{1,2K-1}, F_{1,31}, F_{1,32}, ..., F_{1,3K-1}, ...,$$
$$F_{1,K1}, F_{1,K2}, ..., F_{1,KK-1}\} \oplus \{F_{2,11}, F_{3,11}, ..., F_{K,11}$$
$$F_{2,12}, F_{3,12}, ..., F_{K,12}, ..., F_{2,1K}, F_{3,1K}, ..., F_{K,1K}\}.$$

In the previous process, each MU other than the first MU will update $\frac{1}{K}$ of its cache. After the update, the cache will be equally shared between $K-1$ files totally eliminating file F_1, which reduces the delivery rate for the next delivery time slot.

In a similar fashion, in the next time slot the cache of the remaining MUs is updated by eliminating the demanded file and replacing it with parts of the remaining files. At the tth time slot, an MU would either demand a partially cached file or has a repeated demand for an eliminated file. At each time slot, the cache of the remaining MU is equally split between $K-l$ files, where l is the number of files that were already demanded. Let L be the expected number of files to be demanded once , the expected delivery sum rate for decreasing file popularities would be

$$\begin{aligned} R_{\mathrm{d}}(L) &= \sum_{l=0}^{L-1} \frac{K-l-1}{K-l} N + (K-L)N \qquad (3.72) \\ &= \left(K - \sum_{l=0}^{L-1} \frac{1}{K-l} + (K-L) \right) N \qquad (3.73) \\ &= \left(2K - L - \sum_{C=K-L+1}^{K} \frac{1}{C} \right) N \qquad (3.74) \\ &= (2K - L - (\psi(K+1) - \psi(K-L+1)))\, N, \qquad (3.75) \end{aligned}$$

where ψ is the digamma function, and the expected delivery sum rate if all files are demanded only once is

$$R_{\mathrm{d}}(K) = (K - (\psi(K+1) + \gamma))\, N, \qquad (3.76)$$

where γ is the Euler-Mascheroni constant.

□

Chapter 4

Online Caching with Time-Varying Files Popularities and Uncoded Prefetching

Abstract

[1]Proactive caching shows a great potential to minimize peak download rates by caching popular data, in advance, at the edge. Fast-changing file features, such as fast-changing file popularities and file contents, represent a challenge for proactive caching if cache content update is much slower, which decreases the efficiency and usability of caching. We present a dynamic *coded* caching scheme that updates local user caches and optimizes the use of caching resources. The developed scheme assumes partial knowledge of features variation. The developed scheme is presented for a network with one cache-enabled server, that has a pool of files, communicating with K cache enabled receivers with requests limited to the server's file pool. Asynchronous file delivery is assumed as a result of non-flexible receivers' request timing. We show that the file delivery messages can be used to proactively and constantly update the receivers' finite caches by index-coding the update with delivery messages at no additional rate-cost. We also show that this mechanism reduces the downloaded traffic and can be used to reduce other QoS metrics.

Keywords: Coded proactive caching, dynamic features, index-coding.

[1]This chapter was published as "Dynamic Caching for Files with Rapidly-Varying Features and Content", in *IEEE Transactions on Communications.*

4.1 Introduction

Data communication has increased exponentially over the last decade, which leads to a shift from voice traffic with no delay tolerance to delay-tolerant data traffic [3]. This mandates the development of new communication algorithms that are built around the characteristics of data communications. One of data-communication's most important characteristics is the gap between the time of data creation and the time of users' requests. This gap enables the storage of data along the path between the data source and data consumers. In addition to delay tolerance, data communication is characterized by the fact that the request for the generated data is probabilistic, and messages with the same content are generally sent to more than one receiver at different times. Proactive caching is an effective new technique that stores popular content, before being requested, at different nodes in the network to reduce the peak traffic rate. Proactive caching does not *only* improve the Quality of Service (QoS) for the users by reducing the peak traffic rate, but it reduces the network cost of transmission as well. This is because proactive caching enables the optimization of the use of resources, where the caching process is generally done where the network resources are in abundance.

Recent research works have extensively studied proactive caching in wireless networks [4, 7–11, 13–22, 25, 26, 50, 59, 60]. The system models considered in recent works can be categorized into two main categories in terms of file request/delivery settings, synchronous file request/delivery, and asynchronous request/delivery. Also, it can be categorized into two categories based on the coding strategies used. The first strategy stores files or parts of files in target caches (uncoded caching) [4, 7, 8], while the second strategy encodes files or parts of files together before caching (e.g., index coding [1]), resulting in a coding rate gain. Pioneering the wireless proactive caching research, the authors of [4] proposed adding a cloud storage small-cells to wireless networks as a tool to increase cloud capacity, decrease latency, and decrease the peak rate stressing network resources. In [7], a cache-aided small-cell network with moving users was considered. The proactive content placement of different cells is optimized based on localized content request distribution to minimize the average delay of all users. In [8], the content placement of a small cell network that lacks the request statistics is optimized using reinforcement learning.

In [9], the authors studied proactive caching for a K-user broadcast channel and developed an index-coded caching scheme that exploits the broadcast nature of the channel to minimize the expected delivery rate. They showed that a coded multi-casting gain can be obtained through the availability of caches despite different users' requests. Coded caching differs from other traditional algorithms and uncoded proactive caching by requiring the optimization of the channel use for the placement and delivery phases jointly, where only request statistics is available at the placement phase. The authors of [10, 11] extended the work of [9] to the case of nonuniform requests. The authors of [10] developed a near-optimal scheme for the case of the non uniform requests, while the authors of [11] presented a lower bound on the delivery rate and developed an achievable scheme with a delivery rate that is within a constant factor from the lower bound. The authors of [13] developed an extension to the scheme proposed in [9] to work online, where the user issues a sequence of requests and the cache is continuously updated. They characterize approximately the optimal long-term average rate of the shared link. In contrast, the authors of [14] focused on the case of uncoded placement and derived the exact memory-rate trade-off. Furthermore, other works developed decentralized schemes that can be used for coded caching as in [15, 16]. Coded caching was then extended to study various models, for example, the interference channel as in [17–19], and wireless device-to-device (D2D) network as in [20], [21]. The work in [22, 50] provided more practical applications of coded caching. In [50], the authors proposed a two-hop wireless network for video multicasting using coded caching.

The main challenge for proactive caching is that the cache hit rate is generally low [54], especially when the content library is large. This is a bigger challenge for classical coded caching because it depends on users requesting their files delivered at the same time. However, the low cache hit rate shows that file requests that can be fulfilled from cached contents in a wireless network are generally distant in time. As such, some works relaxed the assumption of synchronous delivery for coded caching. Caching with asynchronous coded delivery was studied in [25, 26, 59, 60]. The authors of [26] studied a network with constant file popularities where users' requests are in the form of a file request with a deadline for delivery. They investigated the case the deadlines for all users are the same, whereas the authors of [25] studied the other case where the users set different deadlines for delivery. They provided two different coding schemes for the case where the requests are known at the placement phase and for the case where the requests are revealed throughout the delivery phase. While index-coded delivery struggles with a low cache hit rate, it can be used to update other user caches. In practice, this can be useful when the file popularities and users' preferences are learned over time or in the case they change over time. Applications with changing file popularity are rarely considered in a coded caching setting despite their prevalence in practice. An example is the effect of breaking news on the popularities of media broadcast content. The authors of [55] studied video streaming request patterns and presented the patterns change over time. They found that users' video interests change throughout the day. This necessitates continuous cache update to follow the changes in file popularities. Fig. 4.1 plots the users' interests of different video genres throughout the day. The authors of [59, 60] studied asynchronous delivery for the condition the server has to fulfill users' requests within one time slots. They assumed that the file popularities are time-variant (within the delivery phase), and assumed that the user requests are not known a priori. The authors of [59] provided a coded scheme for a special case where the popularities of the most popular-files are fading, while the authors of [60] provided a coding scheme for arbitrary file popularities variations. They provided cache update and delivery schemes for asynchronous and mixed (synchronous plus asynchronous) deliveries. They used an offline approach where the file popularities variations are assumed to be accurately predicted/provided to the server at the placement phase. Files with time-varying content is another category of files for which applying proactive caching is challenging. In this category, the content of the same file partially or fully change from time to time. The usability of each file in this category (e.g. sensor data) depends on its rate of change, i.e., on its age of information (AoI) [61]. The authors of [62] studied the relation between delay and data freshness in a cache-aided network. The authors of [61] developed an update policy for a cache-aided server that minimizes the AoI in an uncoded caching setting. Generally, the majority of files have fast-changing metrics/features that the caching process must consider to have efficient utilization of the caching resources. For example, file-specific delay and delay-jitter change as its route get congested. In this

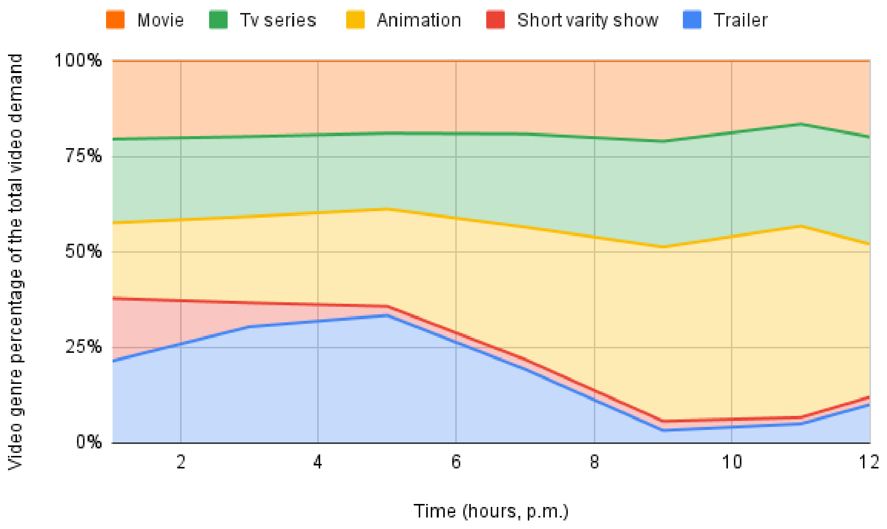


Figure 4.1: Request patterns for videos through daytime.

chapter, in contrast to the offline approach of [59], [60], we develop a *dynamic* code caching scheme that works for the general case of *arbitrary* number of fast-changing multiple features and any predefined *importance metric*. We present the results for the special cases of files with time-varying popularity and time-varying content as well. We study asynchronous coded caching for a network that consists of a server and K receivers with local caches. We assume that the change in features/metrics is known one time slot ahead. We provide a dynamic coded changing scheme that responds to the changes in those features/metrics as information about them is constantly updated at the server. The transmission of information from the server to the receivers occurs over two phases: the placement and delivery phases. The receivers' requests and the delivery of the requested files occur in an *asynchronous* fashion, i.e., in a separate time slot within multiple time slots in the delivery phase. As such, the system model in consideration differs from other coded caching problems considered in the literature that have one time slot delivery phase. The new model tackles a less constrained setting as compared to [13], [14], and [59] as well. In this model, the requests are non-uniform, the distributions of the features are arbitrary, with no constraints on the receivers' requests or delivery times. The proposed scheme optimizes the receivers' cache update to minimize future delivery sum rate. The proposed scheme does not send separate messages that generate additional rate; instead, it uses index coding [1] to send a joint delivery and update messages. The scheme's index coding transmission is formulated as a linear program. Linear programming fixed point methods are leveraged to find the optimal expected delivery sum rate of the scheme. Additionally, the structure of the formulated optimization problem of the scheme is exploited to develop a low complexity optimal solution. Numerical results are used to elaborate on the advantages of the proposed scheme compared to other caching schemes. The key contributions of this work can be summarized as follows:

- We propose a caching scheme for dynamic coded file delivery and caches update. The scheme is designed to take into consideration multiple changing files features and partial knowledge of the features changes.

- We provide a general framework for using the proposed scheme to optimize different dynamic metrics.

- The optimization of the proposed scheme is formulated as a linear program, which enables achieving optimal solutions using efficient linear solvers.

- Based on the characteristics of our optimization problem, we identify a class of linear problems with symmetric constraints' coefficients. A new non-iterative

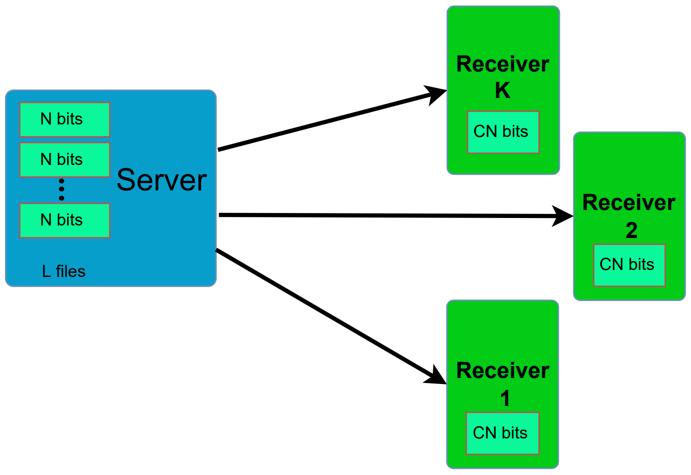


Figure 4.2: Network architecture (e.g. Satellite broadcast network).

algorithm with a linear time complexity that solves any optimization problem belonging to our newly identified class is developed.

Notation: We use calligraphy fonts, i.e., $\mathcal{S}$, to denote sets. card($\mathcal{S}$) to denote its cardinally, We use $\binom{\mathcal{S}}{k}$ to denote the set of all subsets of $\mathcal{S}$ with size k. We use $\oplus$ to denote bit-wise XOR operation and $A\backslash B$ to represent the bits in A that are not in B. We use $O(.)$ to denote the worst case complexity order and Φ to denote an empty set.

4.2 System Model

The system model in consideration consists of a server that communicates with K receivers. The server and receivers are assumed to have limited size caches and are all single antenna nodes, and the receiver cache size is CN bits. The receivers' requests

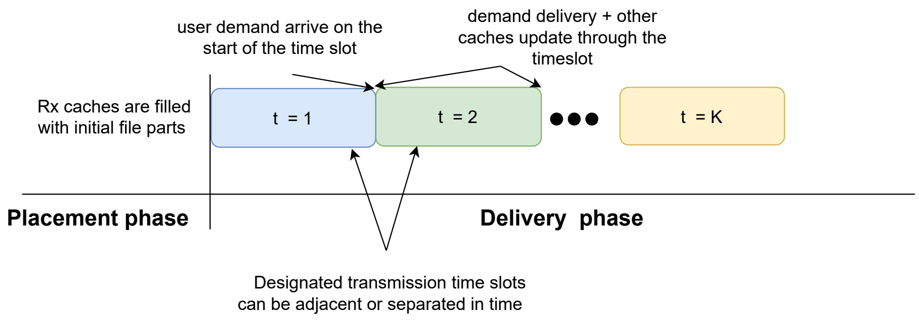

user demand arrive on the start of the time slot
demand delivery + other caches update through the timeslot
Rx caches are filled with initial file parts
t = 1
t = 2
t = K
Placement phase
Delivery phase
Designated transmission time slots can be adjacent or separated in time

Figure 4.3: Placement and delivery phases.

are assumed to be limited to an L-files-library $\mathcal{F} = \{F_1, F_2, ..., F_L\}$, N bits each, that is available at the server. The system model is illustrated by the block diagram shown in Fig. 4.2. An example application of the considered system model is satellite broadcast networks, where the satellite is the server, receivers are the users, and changing file popularities happens over on-demand contents.

The information transfer from the server to the receivers is split into two phases, namely, the placement and delivery phases. At the first (placement) phase, a group of chosen files parts of the library $\mathcal{F}$ is transferred from the server to the receivers' local caches before the receivers' requests arrive at the server. The receivers' requests then arrive at the server through the second (delivery) phase. Each receiver requests one file from the library F_i, $i \in \{1, 2, ..., L\}$, at a separate time slot $t \in \{1, 2, ..., K\}$ of the delivery phase. Any missing files parts belonging to the requested file and that are not available at the requesting receiver's local cache are then sent by the server to the receiver. The server sends the missing parts to the receiver request during the same time slot of the request (Fig. 4.3). Without loss of generality, we assume that the tth receiver request occurs at the tth time-slot.

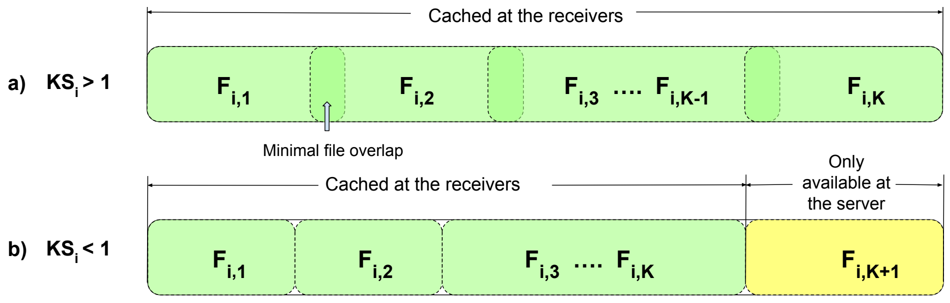


Figure 4.4: Cache placement comparison.

4.2.1 Changing file characteristics

We assume that each file is characterized by G different features that define the conditions of its use and transmission. Weights that define the relative edge of each file are stored in a feature vector $\boldsymbol{b}_{g,t} = [b_{g,1,t} .... b_{g,L,t}], g \in G, t \in K$, where the feature vector changes over time and is file specific. Larger values in the vector correspond to higher priority to be cached. The case of time-varying popularities is an example of file features that change with time. Specifically, the receiver requests are not known a priori, while the probability that a receiver requests a certain file (file popularities) is known one time-slot ahead. The file probabilities vector is a feature vector with weights that differentiate the importance of each file. In the following, we assume that the file popularities are changing over time.

4.2.2 Placement phase

Let $S_{i,1}$ denotes the size, normalized with respect to N, of each part of file F_i chosen to be cached at the placement phase at any receiver, i.e., $\sum_{i=1}^{L} S_{i,1} = C$. While the caching is symmetric in terms of size over all receivers, the content of each part is different. That is to say, each receiver caches a different (possibly overlapping) equally sized part of size $S_{i,1}$, $i = 1, 2, ..., L$, of each file i. Note that parts of different files cached at the same receiver are generally different in size.

4.2.3 Delivery phase

The receivers' requests are sent to the server, each in a separate time slot during the delivery phase. The server sends missing parts of the files requested by respective receivers, that are not available at the requesting receiver. Let $S_{i,t}, t = 2, ..., K$, be the normalized size, at the time slot t, of parts of file F_i that are cached at any receiver. The normalized sizes of the cached parts, $S_{i,t}$, $\forall\, i = 1, 2, ..., L, t = 2, ..., K$, are to be optimized to minimize the expected delivery sum rate. The request of the tth receiver is assumed to arrive at the server at the tth time slot, without loss of generality, i.e., the delivery phase transmission occurs over K time slots. The server delivers the missing parts of the file requested by each receiver by the end of the time-slot of the request. Evidently, the delivery rate depends on both the receivers' requests and the caches' contents of the receivers at the time of request. For the case that the actual requests are not known a priori, we optimize the expected delivery sum-rate rather than the actual delivery sum rate. Since the probability that the ith file is requested at time slot t is $P_{i,t}$, the expected delivery rate of the tth time slot, is

$$R_t = \sum_{i=1}^{L} P_{i,t}(1 - S_{i,t}). \tag{4.1}$$

Our aim is to design the coded cache placement of the coded message of the delivery phase to minimize the expected transmission rate of the delivery phase.

4.2.4 Index coding

In this subsection, we define index coding and the underlining transmission problem it solves. The index coding transmission problem is a situation where a set of W independent messages $\mathcal{M} = \{M_1, M_2, ..., M_W\}$ are available at the transmitter, and we need to send different subsets of them to K receivers. The kth receiver requests a set of messages $M_k \subset \mathcal{M}$, while it already obtained another set of message A_k as side information. A receiving node does not need a message that is already available to it, i.e., $M_k \cap A_k = \Phi$. An index code $\Pi_n(\mathcal{Z}, n, R)$ is used by the transmitter to fulfill the destination needs. The code $\Pi_n(\mathcal{Z}, n, R)$ is composed of a finite alphabet $\mathcal{Z}$ of cardinality $|\mathcal{Z}| > 1$, a joint encoding function, h^c, and a separate decoding function, $h^d_{k,i}$, for the message M_i at the kth receiver. The encoding function h_c maps all the messages to the sequence of transmitted symbols $h^c(M_1, M_2, ..., M_K) = X^n$, where $X^n \in \chi^n$ is the sequence of symbols transmitted over n channel uses. A message $M_k, k \in 1, 2, ..., W$, is a random variable uniformly distributed over the set $F_k \in \{1, 2, ..., |X|\}^{nR_w}$, where $R \in \mathcal{R}^W_+$ is a rate vector $R = (R_1, R_2, ..., R_W)$ in positive real vector space that satisfies the condition that $|X|^{nR_w}$ is an integer for all $1, 2, ..., W$. The decoding function at each receiving node is $g_{k,i}(X_n, A_k) = \hat{M}_{k,i}, \forall i$, where $M_i \in M_k$. An achievable rate tuple $R = (R_1, R_2, ..., R_K) \in R^M_+$ exists if for each $\epsilon, \delta > 0$, there is $(X^n, n, (\bar{R}_1, \bar{R}_2, ..., \bar{R}_K))$ coding scheme, for some X^n, n, such that $\forall w \in 1, 2, ..., W, \bar{R}_w \geq R_w - \delta$.

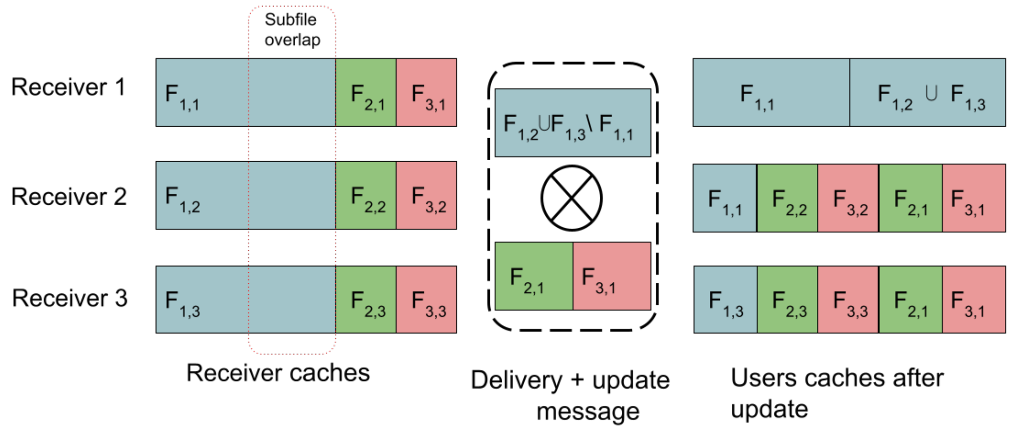
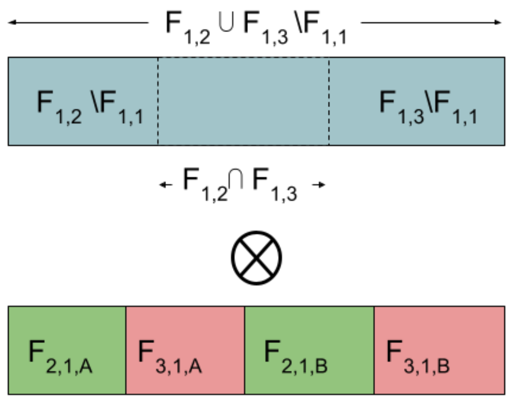

Subfile overlap
Receiver 1
Receiver 2
Receiver 3
$F_{1,1}$ $F_{2,1}$ $F_{3,1}$
$F_{1,2}$ $F_{2,2}$ $F_{3,2}$
$F_{1,3}$ $F_{2,3}$ $F_{3,3}$
Receiver caches
$F_{1,2} \cup F_{1,3} \setminus F_{1,1}$
$\otimes$
$F_{2,1}$ $F_{3,1}$
Delivery + update message
$F_{1,1}$ $F_{1,2} \cup F_{1,3}$
$F_{1,1}$ $F_{2,2}$ $F_{3,2}$ $F_{2,1}$ $F_{3,1}$
$F_{1,3}$ $F_{2,3}$ $F_{3,3}$ $F_{2,1}$ $F_{3,1}$
Users caches after update

Figure 4.5: Examples of cache update message.

4.3 Dynamic coded caching

In this section, we present the proposed dynamic scheme used for coded caching. The objective of the devised dynamic coded caching scheme is to minimize the delivery phase expected sum rate using a limited knowledge of the dynamics of content and features vector changes. To achieve this objective, it uses proactive caching and proactive coded cache update. The caching scheme is similar to the scheme presented in 3. However, caching and updates dynamically change according to the change in file popularities according to the slot by slot acquired knowledge. The caching scheme works to update the receivers' local caches to follow the changes in popularity using the delivery messages sent in the network. In other words, as the set of most popular files is constantly changing, the caching scheme updates the caches to make the caches have more parts of the newly popularized files. Since the scheme leverages the delivery messages sent in the network to encode the required updates, it keeps the caches updated without increasing the delivery phase expected sum rate. Index



$F_{1,2} \cup F_{1,3} \backslash F_{1,1}$
$F_{1,2} \backslash F_{1,1}$
$F_{1,3} \backslash F_{1,1}$
$F_{1,2} \cap F_{1,3}$
$\otimes$
$F_{2,1,A}$
$F_{3,1,A}$
$F_{2,1,B}$
$F_{3,1,B}$

Figure 4.6: Cache plus delivery update message in details with exact bit alignment, delivery message at the top and update message at the bottom.

coding is used to encode the required updates with the delivery messages. The scheme will be reviewed in short for completeness.

4.3.1 Placement phase

Each user, m, caches a part (subfile) $F_{i,m}, m = 1, 2, ..., K$ of size $S_{i,1}$, $0 < S_{i,1} \leq 1$ of each file i such that the overlap between the subfiles of each file is minimal (Fig. 4.4). The value of $S_{i,1}$ is chosen to minimize the expected delivery rate. For small subfiles $KS_{i,1} \leq 1$, the file has $K + 1$ parts as the file is not completely distributed among receivers' caches and the remaining part $F_{i,K+1}$ is only available at the server (Fig. 4.4b). This part has a size of $1 - KS_{i,1}$. While for Large subfiles sizes $KS_{i,1} > 1$, the file is completely cached at the receivers side and the size of $F_{i,K+1}$ is zero (Fig. 4.4a). The motivation for storing different subfiles at different nodes is to help the receivers decode the delivery messages as explained next.

4.3.2 Delivery and update phase

The messages of the delivery phase are designed to deliver the missing part of the requested file and to be used to update the non-requesting receivers caches at each time slot. In that sense, we should note that the continuous update process does not add any rate to the delivery phase sum rate by design. In a given time slot, the BS sends a message for both delivery and update M_{joint}, which is composed of two messages XORed together, M_{delivery} which is intended for the requesting users and M_{update} which is intended for other users. Assume that file F_i is requested by the k receiver at the kth time slot. The server's responding delivery message should include the missing subfiles of F_i at the kth receiver, i.e., the server sends

$$\begin{aligned} M_{\text{delivery}} &= F_i \backslash F_{i,k} && (4.2) \\ &= \{F_{i,1} \cup F_{i,2} ... \cup F_{i,k} \cup F_{i,k+1} \cup F_{i,Q}\} \backslash F_{i,k}, \quad Q = \begin{cases} K, & KS_{i,1} \geq 1, \\ K+1, & KS_{i,1} < 1. \end{cases} && (4.3) \end{aligned}$$

Meanwhile, the server exploits the message to update the other $K-1$ receivers' caches in response to the change in the feature vector. The server needs to send subfiles of files that have an improved feature vector (increase in some of its components values). This has to be at the cost of removing subfiles of files with deteriorated feature vector. To achieve this, the server sends the following update

$$M_{\text{update}} = \{F_{1,k} \cup ... \cup F_{i-1,k} \cup F_{i+1,k} \cup ... \cup F_{L,k}\}. \tag{4.4}$$

As such, the server sends a delivery plus update message

$$M_{\text{joint}} = M_{\text{delivery}} \oplus M_{\text{update}}, \tag{4.5}$$

where the sizes of the content of the M_{update}, $F_{q,k}, q \in L\backslash i$, are optimized to minimize the delivery rate (as explained in Section 4.4). Given the design of the message M_{joint}, the requesting receiver is able to decode it to extract its request $F_i \setminus F_{i,k}$ as it has the subfiles $\{F_{1,k}, F_{i-1,k}, ..., F_{i+1,k}, F_{L,k}\}$ in its caches. Meanwhile, the other receivers can decode their dedicated parts of the message M_{update} as they can partially decode M_{joint} using the parts they have of M_{delivery} as shown in (Figures 4.5 and 4.6). Figure 4.5 shows the delivery plus update message and the receivers cache content after update, while Figure 4.6 shows the exact XOR alignment of different file parts, of the delivery plus update message, for successful decoding. $F_{i,l,A}, F_{i,l,B}$ are used to denote the first and second halves of part $F_{i,l}$, respectively. The slicing of parts $F_{2,1}$ and $F_{3,l}$ along with the arrangement in Fig. 4.6 guarantee a symmetric cache update for the second and the third users. Figure 4.5 shows the components of the delivery message (top middle), which the part of file one that user one needs, and shows the components of the update message (bottom middle) which are parts of the second and the third files distinated for the second and third users for cache update.

Update example: three receivers, three files, one feature is time-varying (file popularities):

Appraise a toy example network that is composed of three receivers and three files, where the file popularities change according to Table 4.1, and files parts are cached at each receiver at the placement phase with the sizes in Table 4.2.

If the first receiver requested the first file at the first time slot, then the server delivery message is composed of $F_1\backslash F_{1,1}$ to be delivered to the first receiver. Meanwhile

Table 4.1: File request probabilities through the delivery phase.

Time	$P_{1,t}$	$P_{2,t}$	$P_{3,t}$
$t = 1$	0.4	0.3	0.3
$t = 2$	0.3	0.4	0.3
$t = 3$	0.3	0.4	0.3

Table 4.2: Placement Cached parts sizes.

Time	$S_{1,1}$	$S_{2,1}$	$S_{3,1}$
Placement	0.4	0.3	0.3

the update message is sent to the other two receivers according to Table 4.3. The corresponding delivery plus update message (Figures 4.5,4.6) is

$$[F_{1,2} \cup F_{1,3}] \oplus [F_{2,1}, F_{3,1}]. \tag{4.6}$$

The delivery rate of the delivery plus update message at the first time slot is equal to the size of the missing part of F_1 at the first receiver or 0.5. The expected delivery rate at the second time slot is $R_2 = P_{2,1}(1-S_{1,2})+P_{2,2}(1-S_{2,2})+P_{2,3}(1-S_{3,2}) = 0.66$. On the other hand, if no cache update is performed, the expected delivery rate of the second time slot would be $R_2 = P_{2,1}(1-S_{1,1})+P_{2,2}(1-S_{2,1})+P_{2,3}(1-S_{3,1}) = 0.74$. The previous results show a reduction of the expected delivery rate at the second time slot due to using the proposed update scheme compared to using traditional caching. It should be noted that the update did not add to the delivery rate of the first time slot. Evidently, the choice of the file parts sizes, $S_{i,j}$, $j = 0, 1, ..., K-1$, $i = 1, 2, ..., L$, is the main factor towards maximum rate reduction. In the following section, the dynamic

Table 4.3: Cached parts sizes after the first update (after the first time slot).

Time	$S_{1,2}$	$S_{2,2}$	$S_{3,2}$
$t = 2$	0.3	0.4	0.3

optimization of the parts size (and the expected delivery sum rate) is presented in detail and the underlying linear optimization is formulated. The minimization is performed for the placement phase and for each time-slot independently.

4.4 Dynamic Delivery Rate Optimization

In this section, the cached files parts size $S_{i,j}, j = 1, ..., K$, are optimized such that the expected delivery sum rate is minimized. The optimization is performed with limited knowledge about the file popularities change. Since the server station has only the next time slot information, the cache content is optimized to minimize the next time slot expected delivery rate each time a new message is sent. The delivery and update problem can be formulated as K linear optimization problems. Each optimization problem depends on the following time slot file popularity. It is expected that the limited information would limit the ability to optimize the cache sizes and predictably increase the delivery rate as compared to performing the optimization for all time slots jointly if all information is available at the placement phase.

4.4.1 Formulation of cache update constraints

In this subsection, we will present the limitations that the finite size of the delivery plus update message impose on the change of receivers caches contents. These limitations are put in the form of linear constraints to the optimization problem of the expected delivery sum rate.

Each part (normalized) size is limited by the size of the respective file. This constraint is represented by

$$0 < S_{i,t} \ \leq \ 1, \ \ \forall t = 1, 2, ..., K, \ \ \forall i = 1, 2, ..., L. \tag{4.7}$$

Since the normalized cache size of each user is C, the sizes of the parts cached at each user are constrained as

$$\sum_{i=1}^{L} S_{i,t} \ = \ C, \ \ \forall t = 1, 2, ..., K, \ \forall i = 1, 2, ..., L. \tag{4.8}$$

The size of the cache update message is limited to the size of the missing part of the requested file $1 - S_{d_t,t}$. Therefore, any update to the size of the cached part of any file can be formulated as

$$S_{i,t+1} - S_{i,t} \ \ \leq 1 - S_{d_t,t}, \ \forall i = 1, 2, ..., L, \ t = 1, 2, ..., K - 1. \tag{4.9}$$

The above message-size constraint $(1 - S_{d_t,t})$ applies to changes in the sizes of any combination of size updates of different files as well, i.e.,

$$\sum_{\mathcal{F}_l \in \mathcal{W}} S_{i,t+1} - S_{i,t} \leq 1 - S_{d_t,t}, \ \forall i = 1, 2, ..., L, \ t = 1, 2, ..., K - 1,$$
$$\mathcal{W} = \binom{\mathcal{F}}{l}, \ \forall l = 2, 3, ..., K. \tag{4.10}$$

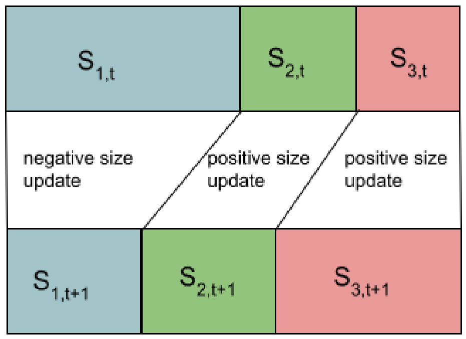


Figure 4.7: Cache size update example, cache division before update at the top and after update at the bottom.

where $\binom{\mathcal{W}}{l}$ denotes the set of all subsets of $\mathcal{F}$ with size l and $\mathcal{F}_l$ denotes a sub-library $\mathcal{F}_l \subseteq \mathcal{F}$ composed of l files. Note that the value of the size update $S_{i,t+1} - S_{i,t}$ can be negative for a particular file i, which corresponds to reducing the size of the cached part of such file to use the space for caching other files (Fig. 4.7), such that the constraint in (4.8) is achieved. In such a case, the update message would not include any parts of file i, which is the case for the first file in the three receivers example explained above. While negative size updates are allowed, they do not translate into more space in the update message for positive updates. They only affect the user cache division. To mathematically establish this distinction, Eq. (4.10) constraints the sum of size updates of any combination of files, in particular, any combination of files with a positive size update is constrained by the update message size. In general, the size updates are split into two groups; files with positive size update (cache content increase) and files with negative size update (cache content reduction). Figure 4.7 explains in detail the sizes update example for the threes receiver given in Fig. 4.5. In particular, the updated cache division of user two, and three, where two files (the second and the third files) experience positive size update and one file (the first file) has a negative size update. The space released by reducing the size of the cached part of the first file is used to accommodate the size increase of the cached parts of the second and third files. We show which files experience a positive size update and which files experience a negative cache update and explain negative updates in detail in Section 4.6.

Given the above scheme, the receivers receiving a cache update have limited ability decoding the update message. Each receiver can decode part of the update message. This is a result of the fact that each receiver has part of the delivery message, M_{delivery}, which is the cached part of size , $S_{d_{t+1},t}$, of the requested file. Specifically, if a receiver, t, requested F_i an a receiver, $k \neq j$, has a subfile $F_{i,k}$, receiver k can decode the part of the update message that is XORed with $F_{i,k}$. In other words, the cache *update* of each receiver is limited such that the *update* in file F_i at instant t (i.e., $S_{i,t+1} - S_{i,t}$) is smaller than or equal to $S_{d_{t+1},t}$. This can be represented as

$$S_{i,t+1} - S_{i,t} \;\; \leq S_{d_{t+1},t}, \; \forall i = 1, 2, ..., L, \; t = 2, 3, ..., K-1. \tag{4.11}$$

Similarly, the sum of any combination of the updates is smaller than the decoded part of the update message, which can be formulated as

$$\sum_{\mathcal{F}_l \in \mathcal{W}} S_{i,t+1} - S_{i,t} \leq S_{d_{t+1},t}, \forall i = 1, 2, ..., L, t = 1, 2, ..., K-1,$$
$$\mathcal{W} = \binom{\mathcal{F}}{l}, \; \forall l = 2, 3, ..., L. \tag{4.12}$$

The Cache updates in (4.11) and (4.12) can be negative as explained above. Finally, at an instant t, the receiver requesting file l has no more than $S_{i,t}$ of file F_i. Therefore, the update cannot contain more than $S_{i,t}$ of $F_i, i = 1, 2, ..., L, i \neq l$, XORed with the missing part of F_l such that the receiver requesting F_l be able to decode the update message and extract the missing part $F_l \backslash F_{l,t}$. Hence, the update of any subfile at the other receivers' caches can be constrained as

$$S_{i,t+1} - S_{i,t} \ \leq S_{i,t} \ \forall \ i = 1, 2, ..., K-1, \ t = 2, 3, ..., K-1. \tag{4.13}$$

4.4.2 Rate optimization

Given the above constraints, the optimization problem to minimize the next time slot expected delivery rate can be formulated as

$$\begin{aligned}
\min_{\mathcal{S}^{t+1}} \quad & R_{t+1} = \sum_{i=1}^{L} P_{i,t+1}(1 - S_{i,t+1}), \quad \forall t = 1, 2, ..., K-1, \\
\text{subject to} \quad & S_{i,t+1} - S_{i,t} \leq S_{d_t,t}, \qquad \forall t = 1, 2, ..., K-1, \qquad \forall i = 1, 2, ..., L, \\
& \sum_{\mathcal{F}_l \in \mathcal{W}} S_{i,t+1} - S_{i,t} \leq S_{d_t,t}, \qquad \forall t = 1, 2, ..., K-1, \mathcal{W} = \binom{\mathcal{F}}{l} \ \forall l = 2, ..., L, \\
& S_{i,t+1} - S_{i,t} \leq 1 - S_{d_t,t}, \qquad \forall t = 1, 2, ..., K-1, \qquad \forall i = 1, 2, ..., L, \\
& \sum_{\mathcal{F}_l \in \mathcal{W}} S_{i,t+1} - S_{i,t} \leq 1 - S_{d_t,t}, \ \forall t = 1, 2, ..., K, \mathcal{W} = \binom{\mathcal{F}}{l} \ \forall l = 2, 3, ..., L, \\
& 0 \leq S_{i,t+1} \leq 2S_{i,t}, \qquad \forall t = 1, 2, ..., K-1, \qquad \forall i = 1, 2, ..., L, \\
& \sum_{F_i \in \mathcal{F}} S_{i,t} = C, \qquad \forall t = 1, 2, ..., K,
\end{aligned} \tag{4.14}$$

where $\mathcal{S}^{t+1}$ is the set of files parts sizes cached at $t+1$, i.e., $\mathcal{S}^{t+1} = \{S_{i,t+1}, i = 1, 2, ..., L\}$. The optimization problem (4.14) is a linear programming optimization and can be solved using linear programming methods [56]. Note that the optimal solution of the previous minimization problem is an achievable rate for the system model in consideration. The opitmality of the proposed scheme is yet to be proven. This problem is solved $(K-1)$ times at $(t = 1, 2, ..., K-1)$. On the other hand, at the placement phase the files are cached in accordance with the initial file popularity.

This results in $S_{i,1} = P_{i,1}, \;\; \forall i = 1, 2, ..., L$.

4.5 General Dynamic Optimization

In this subsection, we provide a general framework for using the same scheme to optimize different dynamic metrics. The framework is valid as long as the relation is linear and has a similar structure.

Other metrics $\boldsymbol{b}_{g,t}, \forall g \in 1, .., G, t \in 1, ..., K$ or combinations of metrics can be used to establish the file importance and drive updates. A general optimization of the weighted sum of those metrics and the associated trade-off can be represented as

$$\begin{aligned}
\min_{\mathcal{S}^{t+1}} \quad & \Omega^{t+1} = \sum_{g=1}^{G} v_g \sum_{i=1}^{L} b_{g,i,t+1}(1 - S_{i,t+1}), \quad \forall t = 1, 2, 3, ..., K-1, \\
\text{subject to} \quad & (4.8), (4.9), (4.10), (4.11), (4.12), \text{and } (4.13) \\
& \sum_{\mathcal{W}} S_{i,t+1} - S_{i,t} \leq R_b, \mathcal{W} = \binom{\mathcal{F}}{l} \;\; \forall l = 2, 3, ..., L, \qquad (4.15) \\
& B_{g,t+1}^{\min} < \sum_{i}^{L} b_{g,i,t+1}(1 - S_{i,t+1}) \leq B_{g,t+1}^{\max}, \forall t = 1, 2, 3, ..., K-1 \qquad (4.16)
\end{aligned}$$

where G is the total number of metrics used, $B_{g,t+1}^{\min}$, and $B_{g,t}^{\max}$ are the constraints on the metric $b_{g,t}$, v_g is the weight of feature g, and $R_b = \min(S_{d_t,t}, 1 - S_{d_t,t})$.

4.6 Linear Dynamic Caching (LDC) Algorithm

The dynamic caching algorithm developed in the previous section requires less information to operate compared to the offline algorithm presented in chapter 3. Additionally, the optimization is less complex and faster to solve. To reduce the complexity even further, we propose a linear time solver for the proposed dynamic

caching scheme. In this non-iterative algorithm, the linearity of the problem is exploited to develop a low-complexity algorithm when compared to using traditional linear programming methods. The algorithms are explained in the terms of the changing file popularities case. The proposed algorithm focuses on minimizing the terms of the expected delivery sum-rate that have the largest coefficients in a descending order. In other words, it starts with increasing the sizes of the cached parts of files with the highest probability. However, due to the constraints imposed on the size of decoded update information, the aggregate cache increase, η, of these files is constrained at time t as $\eta = \min(S_{d_t,t}, 1 - S_{d_t,t})$.
The basic idea of the LDC algorithm is to distribute the total possible cache increase η among files in descending order of their popularity, where the cached part of the file with the highest request probability is increased first to the maximum size allowed by its specific constraints. Then, the file of next highest request probability is increased in the same fashion until the caches increase η is depleted. On the other hand, since the receiver cache size is limited, the size of the cached parts the files with the lowest request probability is decreased with an amount equal to the total cache increase η, namely, the total cache decrease. Such decrease is undertaken in ascending order of the file popularities starting by the least probable file until the cache decrease is satisfied.

The proposed low-complexity coded caching algorithm (LDC) can be formally expressed as follows:

Algorithm 1 Algorithm: LDC

```
1. Calculate $\eta = \min(1 - S_{d_t,t}, S_{d_t,t})$.
2. Maximize the sizes of the cached parts of the files with the highest probabilities
   in turn, until respective constraints are met with equality.
3. Stop when the total cache size increase of the files with the highest probability
   is equal to $\eta$.
4. Decrease the sizes of the files with lowest probability in turn (negative size
   updates).
5. Stop when the total cache decrease is equal to $\eta$.
```

Theorem 5. *For the proposed cache update scheme, the LDC algorithm is optimal in the sense that it minimizes the expected sum rate of* (4.14)*.*

proof: The objective function of the expected sum rate in (4.14) can be rewritten as

$$R_{t+1} = 1 - \sum_{i=1}^{L} P_{i,t+1} S_{i,t+1}. \tag{4.17}$$

In mathematical terms, in order to minimize the expected delivery rate, we need to maximize the expected number of bits retrieved from the cache, i.e., $\sum_{i=1,t=1}^{i=L,t=K} P_{i,t} S_{i,t}$. The algorithm identifies the most significant terms or variables and the most significant constraints on each variable, then it maximizes these terms or variables in turn. In the following part of this subsection, we show that transforming the optimization problem into solving a sequence of partial maximization problems is optimal. In other words, we show that ranking the maximization terms and the constraints according to the file popularity is optimal.

4.6.1 Proof of optimality

We first start with a three receivers proof to elaborate on the idea of the proof and then provide the K receiver proof thereafter. The three receivers proof is composed of two parts. In the first part, the optimization is solved graphically for different cases depending on the values of the constraints. An optimal solution is derived graphically for each case. In the second part, it is shown that the optimal solution can be described as a solution of a corresponding system of equations and a simple algorithm is developed to find those solutions. Let $U_{i,t}$ be the size update (increase or decrease) of file F_i given by $U_{i,t} = S_{i,t+1} - S_{i,t}$. The proof key strategy is as follows:

- First, it starts with an assumption that a subset of the updates $\mathcal{U}^- = U_{i,t}, i \in 1, 2, ..., K$, has negative values and show the conditions under which this subset has negative values. This assumption is a direct result of (4.8), as the updates at any time slot sums to zero.

- It then shows that the constraints on the set of non-negative updates can be reduced to a smaller number of constraints. These constraints can be ranked and solved directly (non-iteratively).

- Finally, the proof shows that the argument holds for any number of negative updates smaller than K, which is the maximum possible number of negative updates.

- The strategy is explained for three receivers first, and the generalized for any

K.

By combining (4.9) and (4.11) into one constraint and combining (4.10) and (4.12) into one constraint as well, the constraints (4.8)-(4.13) can be rewritten as

$$\begin{aligned} S_{i,t+1} - S_{i,t} \leq\ & \min(1 - S_{d_t,t}, S_{d_t,t}), && \forall t = 1,2,...,K-1,\ \forall i = 1,2,...,L, && (4.18)\\ \sum_{\mathcal{F}_l \in W} S_{i,t+1} - S_{i,t} \leq\ & \min(1 - S_{d_t,t}, S_{d_t,t}), && \forall t = 1,2,...,K-1,\ \mathcal{W} = \binom{\mathcal{F}}{l}, && (4.19)\\ & && \forall l = 2,3,...,L, && (4.20)\\ 0 \leq S_{i,t+1} \leq\ & 2S_{i,t}, && \forall t = 1,2,...,K-1,\ \forall i = 1,2,...,L, && (4.21)\\ \sum_{i=1}^{L} S_{i,t} =\ & C, && \forall t = 1,2,...,K \quad \forall i = 1,2,...,L. && (4.22) \end{aligned}$$

Moreover, by putting $U_{i,t} = S_{i,t+1} - S_{i,t}$, and $\eta = \min(1 - S_{d_t,t}, S_{d_t,t})$ in the minimization problem (4.14) and its constraints (4.18)-(4.22), it can be rewritten as

$$\begin{aligned} \min_{\mathcal{U}^t} \quad & R_{\text{on}}^{t+1} = \sum_{i=1}^{L} P_{i,t+1}(1 - U_{i,t}) - \sum_{i=1}^{L} P_{t+1,d_{(t+1)}} S_{i,t}, \quad \forall t = 1,2,3,...,K-1,\\ \text{subject to} \quad & U_{i,t} \leq \eta, \qquad \forall t = 1,2,3,...,K-1,\ \ \forall i = 1,2,3,...,L,\\ & \sum_{\mathcal{F}_l \in W} U_{i,t} \leq \eta, \qquad \forall t = 1,2,3,...,K-1,\ \ \forall i = 1,2,3,...,L,\\ & -S_{i,t} \leq U_{i,t} \leq S_{i,t}, \qquad \forall t = 1,2,3,...,K-1,\ \ \forall i = 1,2,3,...,L,\\ & \sum_{i=1}^{L} U_{i,t} = 0, \qquad \forall t = 1,2,3,...,K-1,\ \ \forall i = 1,2,3,...,L, \qquad (4.23) \end{aligned}$$

where the second term of the minimization problem is a constant and can be safely dropped from the objective function. Let $\{U_{1,t}, U_{2,t}, ..., U_{K,t}\} = \{U^*_{1,t}, U^*_{2,t}, ..., U^*_{K}\}$ be the optimal solution of the previous optimization problem. Since each $U_{l,t},\ l \in L, t \in K$ can be negative, and due to the structure of the constraints, where the RHS of all the constraints are equal and all coefficients are equal, all constraints that include one or more negative $U_{l,t}$ will be inactive. This is because each of these constraints has a "dual" with an equal RHS and includes all other $U_{i,t}, i \in K, i \neq l$, except for $U_{l,t},\ l \in K$, (which are negative). For example if

$$U_{1,t} \leq 1, \tag{4.24}$$

$$U_{1,t} + U_{2,t} \leq 1, \tag{4.25}$$

and if $U_{2,t}$ is negative, then,

$$U_{1,t} \leq 1, \tag{4.26}$$

$$U_{1,t} \leq 1 + |U_{2,t}|. \tag{4.27}$$

In the following, we provide an example of three receivers, and three files system to clarify the concept. Afterwards, we discuss the general case of K receivers.

1) <u>**Example: (K=3)**</u> For the updates of a three receivers three files system, at time slot t, the constraints of (4.23) can be written as

$$U_{1,t} \leq \eta, U_{2,t} \leq \eta, U_{3,t} \leq \eta, \tag{4.28}$$

$$U_{1,t} + U_{2,t} \leq \eta, \tag{4.29}$$

$$U_{2,t} + U_{3,t} \leq \eta, \tag{4.30}$$

$$U_{1,t} + U_{3,t} \leq \eta, \tag{4.31}$$

$$U_{1,t} + U_{2,t} + U^*_{3,t} \leq \eta. \tag{4.32}$$

$$-S_{1,t} \leq U_{1,t} \leq S_{1,t}, \tag{4.33}$$

$$-S_{2,t} \leq U_{2,t} \leq S_{2,t}, \tag{4.34}$$

$$-S_{2,t} \leq U_{3,t} \leq S_{3,t}, \tag{4.35}$$

$$\sum_{i=1}^{3} U_i = 0. \tag{4.36}$$

The optimal solution for this problem can be classified into two categories. The first is when only one update has a negative value and the second is when two updates have a negative value.

Category one: one negative update

For an optimal solution that belongs to the first category, assume without loss of generality that $U^*_{3,t}$ has a negative value, while the other updates have a positive value. The constraint in (4.31), can be eliminated as it is inactive due to the existence of (4.28). Similarly, (4.30) and (4.32) can be eliminated because of (4.28) and (4.29), respectively. Moreover, (4.28) and (4.35) are inactive given that $U^*_{3,t}$ has a negative value and η is greater than or equal to zero. Further, since the variables $U_{1,t}$ and $U_{2,t}$ are guaranteed to have positive values by the definition that only $U^*_{3,t}$ has a negative value, (4.28) and (4.28) are redundant because of (4.29). As a result, The update message size constraints (4.28) to (4.36) are further reduced to

$$\begin{aligned} U_{i,t} &\leq S_{i,t} \quad \forall i = 1, 2, \;\; \forall t = 1, 2, ..., K, & (4.37)\\ |U^*_{3,t}| &\leq S_{3,t} \quad \forall t = 1, 2, ..., K, & (4.38)\\ U_{1,t} + U_{2,t} &\leq S_{1,t} + S_{2,t}, & (4.39)\\ U_{1,t} + U_{2,t} &\leq \eta, & (4.40)\\ U_{1,t} + U_{2,t} &= |U^*_{3,t}|, & (4.41) \end{aligned}$$

where (4.39) is the sum of the two inequalities in (4.37). Investigating (4.39), (4.40), and (4.41), $|U^*_{3,t}|$ is guaranteed not to be larger than $\min(\eta, S_{1,t} + S_{2,t})$. Hence, for a given value of $|U^*_{3,t}|$, the constraints (4.37) to (4.41) are reduced to

$$U_{i,t} \leq S_{i,t} \quad \forall i = 1, 2, \tag{4.42}$$

$$U_{1,t} + U_{2,t} = |U^*_{3,t}|, \tag{4.43}$$

and the solution to the optimization problem in this case can be explained as follows. Given that the coefficients of the variables in the update message size constraints (4.42) and (4.43) are equal to unity, $U_{i,t}, i = 1, 2$, should be maximized until the constraints are satisfied. Given that we established the structure of the solution, it remains to show that maximizing U_1, U_2 in order of the respective file popularity is optimal. The objective function of the optimization problem (4.23) can be rewritten as

$$R^{t+1}_{\text{on}} = c_o - \sum_{i=1}^{K} P_{i,t+1} U_{i,t}, \quad \forall t = 1, 2, 3, ..., K-1, \tag{4.44}$$

where c_o is a constant that equals $\sum_{i=1}^{L} P_{i,t+1} - P_{t+1,d_{(t+1)}} S_{i,t}$. Let $P_{1,t+1} > P_{2,t+1}$, the optimal solution depends on the relation between the values of $S_{1,t}, S_{2,t}$, and $|U^*_{3,t}|$. There are four different cases for how the constraints (4.42) and (4.43) relate to each other depending on the values of $S_{1,t}, S_{2,t}$, and $|U^*_{3,t}|$.

Through the second part of the proof, we will solve each of the four cases using the graphical method [63], then show that the optimal solution is the same as the LDC algorithm's solution for all cases. Figure 4.8 shows the four cases and graphically represents the solution in each case. Graphical method draws the constrains and the corresponding feasible region to obtain an optimal solution. The feasible region (bounded in our case) is the set of all points that satisfies the constraints and is the convex hull outlined by the constraints. For a linear program, one (or more) of the extreme (corner) points are optimal solution(s). In Fig. 4.8, the first two constraints in (4.42) are plotted in dashed black lines, while the third constraint (4.43) is plotted in red solid line. The four subplots represent the four different cases for the value of $|U^*_{3,t}|$ relative to $S_{1,t}, S_{2,t}$ or the relation between the first two constraints on one side and the third constraint on the other side.

Let $\Xi_I, \Xi_{II}, \Xi_{III}$, and Ξ_{IV} be the values of the expected total update size $\Xi = \sum_{i=1}^{K} P_{i,t+1} U_{i,t}$ at points I, II, III, and IV of Fig. 4.8, respectively. The point that achieves the largest value of Ξ corresponds to the point of smallest value of R^{t+1}_{on} and it represents the optimal solution,i.e.,

$$R^{t+1*}_{\text{on}} = c_o - \max(\Xi_I, \Xi_{II}, \Xi_{III}, \Xi_{IV}). \tag{4.45}$$

Next, we analyze the resultant four cases for the constraints (4.42) and (4.43) and graphically find the optimal solution of (4.23) for each case. We then prove that the LDC algorithm reaches the same solutions.

- Case 1: $|U^*_{3,t}| \geq S_{1,t}, S_{2,t}$ (Fig. 4.8(a)).

  Given that the values of $U_{1,t}$ and $U_{2,t}$ at points I, II, III and IV are

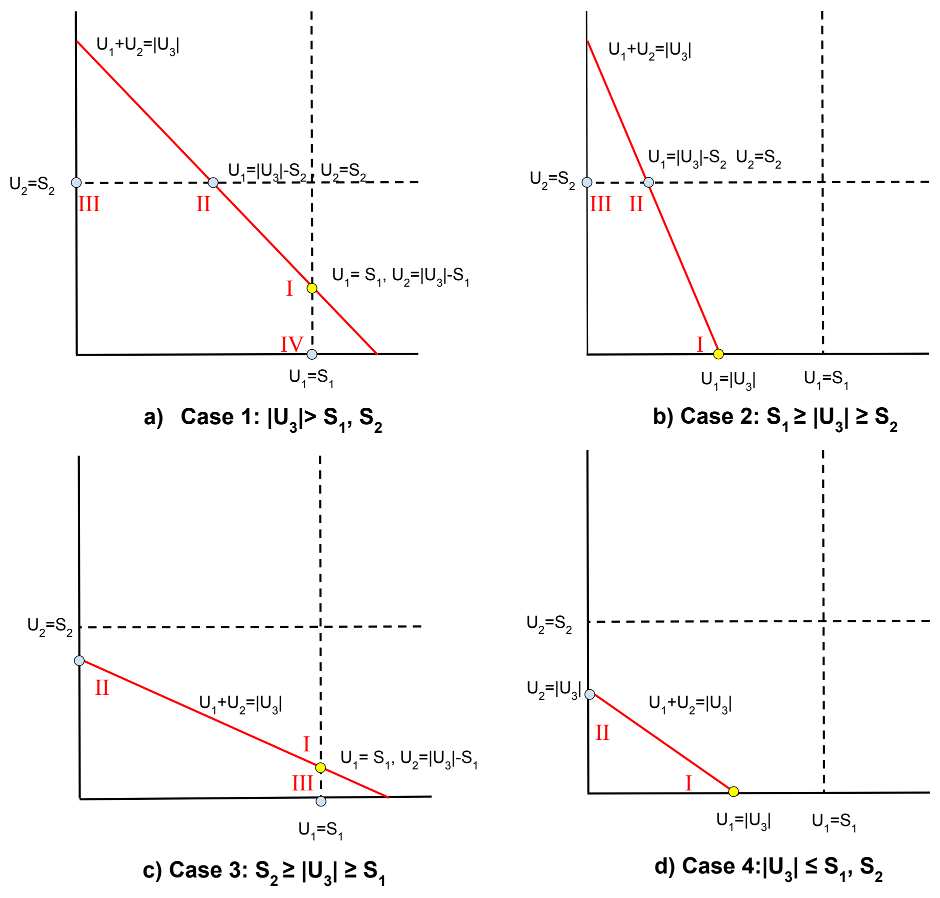
$$I: \qquad U_{1,t} = S_{1,t}, \qquad U_{2,t} = (|U^*_{3,t}| - S_{1,t}), \tag{4.46}$$

$$II: \qquad U_{1,t} = (|U^*_{3,t}| - S_{2,t}), \quad U_{2,t} = S_{2,t}, \tag{4.47}$$

$$III: \qquad U_{1,t} = 0, \qquad U_{2,t} = S_{2,t}, \tag{4.48}$$

$$IV: \qquad U_{1,t} = S_{2,t}, \qquad U_{2,t} = 0, \tag{4.49}$$

  the corresponding average update sizes would be as follows



$U_1+U_2=|U_3|$
$U_2=S_2$
$U_1=|U_3|-S_2$, $U_2=S_2$
$U_1=S_1$, $U_2=|U_3|-S_1$
I
II
III
IV
$U_1=S_1$
a) Case 1: $|U_3|> S_1, S_2$
$U_1+U_2=|U_3|$
$U_2=S_2$
$U_1=|U_3|-S_2$ $U_2=S_2$
III II
I
$U_1=|U_3|$
$U_1=S_1$
b) Case 2: $S_1 \geq |U_3| \geq S_2$
$U_2=S_2$
II
$U_1+U_2=|U_3|$
I
$U_1=S_1$, $U_2=|U_3|-S_1$
III
$U_1=S_1$
c) Case 3: $S_2 \geq |U_3| \geq S_1$
$U_2=S_2$
$U_2=|U_3|$
$U_1+U_2=|U_3|$
II
I
$U_1=|U_3|$
$U_1=S_1$
d) Case 4: $|U_3| \leq S_1, S_2$

Figure 4.8: Graphical solution for the optimization problem (4.23) with three receivers, three files: four different cases of the constraints.

$$\Xi_I = P_{1,t}S^t_{1,t} + P_{2,t}(|U^*_{3,t}| - S_{1,t}), \tag{4.50}$$

$$\Xi_{II} = P_{1,t}(|U^*_{3,t}| - S_{2,t}) + P_{2,t}S_{2,t}, \tag{4.51}$$

$$\Xi_{III} = P_{2,t}S_{2,t}, \tag{4.52}$$

$$\Xi_{IV} = P_{1,t}(|U^*_{3,t}| - S_{2,t}). \tag{4.53}$$

Since $\Xi_{II} > \Xi_{III}$ and $\Xi_I > \Xi_{IV}$, the points III and IV are sub-optimal. Given that

$$\Xi_I - \Xi_{II} = (P_{1,t} - P_{2,t})((S_{1,t} + S_{2,t}) - |U^*_{3,t}|), \tag{4.54}$$

and since (4.42) and (4.43) constrain $|U^*_{3,t}|$, i.e., $|U^*_{3,t}| \le S_{1,t} + S_{2,t}$, $\Xi_I \ge \Xi_{II}$ and the optimal point that achieves the minimum R^{t+1}_{on} is $I = \{S_{1,t}, |U^*_{3,t}| - S_{1,t}\}$. The reason the point I is the optimal solution can be warranted to the fact that $P_{1,t} > P_{2,t}$ in (4.50) and (4.51). The optimal rate can be alternatively obtained by distributing the value of $|U^*_{3,t}|$ over $U_{1,t}$ and $U_{2,t}$ in a two-step maximization. First, maximize the size of the update with highest probability ($U_{1,t} = S_{1,t}$), then maximize the size of the second one $U_{2,t} = (|U^*_{3,t}| - U_{1,t})$.

- Case 2: $S_{1,t} \ge |U^*_{3,t}| \ge S_{2,t}$ (Fig. 4.8(b)).

  The average update sizes at points I, II, III, and IV are

$$\Xi_I = P_{1,t}|U^*_{3,t}|, \tag{4.55}$$

$$\Xi_{II} = P_{1,t}(|U^*_{3,t}| - S_{2,t}) + P_{2,t}S_{2,t}, \tag{4.56}$$

$$\Xi_{III} = P_{2,t}S_{2,t}, \tag{4.57}$$

$$\Xi_I - \Xi_{II} = P_{1,t}|U^*_{3,t}| - (P_{1,t} - P_{2,t})S_{2,t}. \tag{4.58}$$

Since $\Xi_I \geq \{\Xi_{II}, \Xi_{III}\}$, the solution is the point $I = \{|U^*_{3,t}|, 0\}$. Similar to the previous case, the fact that $P_{1,t} > P_{2,t}$ is the reason why the point I is the optimal solution. Similarly, the rate can be obtained by distributing the value of $|U^*_{3,t}|$ over the other two updates using two-step maximization of the updates in the order of their respective probability.

- Case 3: $S_{2,t} \geq |U^*_{3,t}| \geq S_{1,t}$ (Fig. 4.8(c)).

$$\begin{aligned}\Xi_I &= P_{1,t}S_{1,t} + P_{2,t}(|U^*_{3,t}| - S_{1,t}), && (4.59)\\ \Xi_{II} &= P_{2,t}|U^*_{3,t}|, \qquad \Xi_{III} = P_{1,t}S_{1,t}, && (4.60)\\ \Xi_I - \Xi_{II} &= (P_{1,t} - P_{2,t})S_{1,t}. && (4.61)\end{aligned}$$

Since $\{\Xi_I \geq \Xi_{II}, \Xi_{III}\}$, the solution is the point $I = \{S_{1,t}, |U^*_{3,t}| - S_{1,t}\}$. Similar to the two previous cases, the fact that $P_{1,t} > P_{2,t}$ is the reason for the rate optimality of point I and the rate can be obtained in a similar fashion.

- Case 4: $|U^*_{3,t}| \leq S_{1,t}, S_{2,t}$ (Fig. 4.8(d)).

$$\Xi_I = P_{1,t}|U^*_{3,t}|, \quad \Xi_{II} = P_{2,t}|U^*_{3,t}|. \qquad (4.62)$$

Since $\Xi_I \geq \Xi_{II}$, the point $I = \{|U^*_{3,t}|, 0\}$ is an optimal solution. Similar to the three previous cases, the fact that $P_{1,t} > P_{2,t}$ is the reason of the rate optimality of point I and the rate can be obtained in a similar fashion.

That said, the solutions of the four cases can be combined as

$$\begin{aligned}U_{1,t} &= \min(S_{1,t}, |U^*_{3,t}|), && (4.63)\\ U_{2,t} &= \max(0, |U^*_{3,t}| - S_{1,t}). && (4.64)\end{aligned}$$

From (4.38), (4.40), and (4.41), the optimal value for the third update is $|U^*_{3,t}| = \min(S_{3,t}, \eta)$. Therefore, the solution can be rewritten as

$$\begin{aligned} U_{1,t} &= \min(S_{1,t}, \min(S_{3,t}, \eta)), && (4.65) \\ U_{2,t} &= \max(0, \min(S_{3,t}, \eta) - S_{1,t}). && (4.66) \end{aligned}$$

The previous solution assumed that U_3 is negative, so it remains to show the conditions under which an update is negative or positive. Let $P_{1,t+1} \geq P_{2,t+1} \geq P_{3,t+1}$. Given that $U_{1,t} + U_{2,t} + U_{3,t} = 0$, the delivery rate can be rewritten as

$$\begin{aligned} R^{t+1}_{\text{on}} &= c_o - (P_{1,t+1}U_{1,t} + P_{2,t+1}U_{2,t} + P_{3,t+1}U_{3,t}) \\ &= c_o - (P_{1,t+1} - P_{2,t+1})U_{1,t} + (P_{2,t+1} - P_{3,t+1})U_{3,t}, && (4.67) \end{aligned}$$

where c_o is a constant. Since $P_{2,t+1} > P_{3,t+1}$, the previous equation is minimized if U_3 is negative, where U_3 belongs to the file with the lowest request probability. The update $U^*_{3,t}$ will be exclusively negative in two cases. The first case is when $\eta < S_{3,t}$, and the second case is when $\eta \geq S_{3,t} \geq S_{1,t}$. For the first case, the optimal value of the third update would be $|U_3| = \eta$. Since $U_{1,t}$ can not be increased beyond η, then $U_{2,t}$ can not be negative because of the sum constraint (4.41). The solution in this case can be represented as

$$|U^*_{3,t}| = \eta, \qquad (4.68)$$

$$U_{1,t} = \min(S_{1,t}, \eta), \qquad (4.69)$$

$$U_{2,t} = \max(0, \eta - S_{1,t}). \qquad (4.70)$$

A simple version of Algorithm (1) that is inspired by the two-step maximization method explained above can be written to find the optimal updates sizes defined by (4.68), (4.69), and (4.70) for the case that $\eta < S_{3,t}$ as follows.

Algorithm 1.1: special case of Algorithm 1 for $\eta < S_{3,t}$, 3 receivers

---

```
1. Calculate $\eta$, set $\Gamma^+ = \eta$.
2. Set $U_{1,t} = \min(S_{1,t}, \eta))$, and $\Gamma^+ = \eta - U_{1,t}$.
3. Set $U_{2,t} = \Gamma^+$, and $\Gamma^+ = 0$.
4. Set $U^*_{3,t} = -\min(\eta, S_{3,t})$.
```

The second case for which $U^*_{3,t}$ is exclusively negative is when $\eta \geq S_{3,t} \geq S_{1,t}$. Similar to the previous case, R^{t+1}_{on} is minimized if $U_{3,t}$ is negative (4.67). The optimal value of the third update is $|U_3| = S_{3,t}$. Since $S_{1,t} \leq S_{3,t}$, and the first update is constrained as $U_{1,t} \leq S_{1,t}$, $U_{2,t}$ can not be negative due to the sum constraint (4.41). The solution in this case can be represented as

$$|U^*_{3,t}| = S_{3,t}, \tag{4.71}$$

$$U_{1,t} = S_{1,t} \tag{4.72}$$

$$U_{2,t} = S_{3,t} - S_{1,t}. \tag{4.73}$$

Another version of Algorithm 1 that can find the optimal updates sizes defined by (4.71), (4.72), and (4.73) for the case that $\eta \geq S_{3,t}$ is as follows.

Algorithm 1.2: special case of Algorithm 1 for $\eta \geq S_{3,t}$, 3 receivers

```
1. Calculate $\eta$, set $\Gamma^+ = \eta$.
2. Set $U_{1,t} = \min(S_{1,t}, \eta))$, and $\Gamma^+ = \eta - U_{1,t}$.
3. Set $U_{2,t} = \Gamma^+$, and $\Gamma^+ = 0$.
4. Set $U_{3,t}^* = -\min(\eta, S_{3,t})$, and $\Gamma^- = -\eta + U_{3,t}^*$.
5. Set $U_{2,t} = U_{2,t} + \Gamma^-$, and $\Gamma^- = 0$.
```

Category two: two negative updates

Similarly, for the second group of the optimal solutions in which two updates have negative values, assume that $U_{2,t}^*$ and $U_{3,t}^*$ have negative values. All inequality constraints would be inactive except for (4.28) and (4.33). The optimal solution would be $U_{1,t} \leq \min(S_{1,t}, |U_{2,t}^* + U_{3,t}^*|)$. For this case a direct extension of the Algorithm 1.2 can find the optimal solution.

2) **General K receivers:** In the following, the three receivers proof is extended to the K receivers case. For a K receivers network, we have the following constraints

$$\sum_{\mathcal{V}_l} U_{i,t} \leq \eta, \quad \mathcal{V}_l = \binom{\mathcal{F}}{l}, \quad \forall l = 1, 2, ..., L, \tag{4.74}$$

$$U_{i,t} \leq S_{i,t}, \quad \forall i = 1, 2, ..., L, \tag{4.75}$$

$$\sum_{1}^{K} U_{i,t} = 0. \tag{4.76}$$

The all zeros solution is not valid because the popularities are changing. Additionally, one can notice from (4.76) that the optimal solution is composed of a set of negative updates $\mathcal{U}^- = \{U_{i,t}^-, i = 1, 2, ..., R\}$ and a set of non negative updates $\mathcal{U}^+ = \{U_{i,t}^+, i =$

$1, 2, ..., Q\}$, where

$$\sum_{1}^{Q} U_{i,t}^{+} + \sum_{1}^{R} U_{i}^{-} \;\; = \;\; 0. \tag{4.77}$$

Let $S_{i,t}^{+}$ be the normalized cache size of the ith file that has a positive update, then the constraints can be redefined as

$$\begin{aligned} \sum_{\mathcal{V}_l} U_{i,t}^{+} &\leq \eta, \quad \mathcal{V}_l = \binom{\mathcal{U}^{+}}{l}, \; \forall\, l = 1, 2, ..., Q, && (4.78) \\ U_{i,t}^{+} &\leq S_{i,t}^{+}, \quad \forall i = 1, 2, ..., Q, && (4.79) \\ \sum_{1}^{Q} U_{i,t}^{+} &= \sum_{1}^{R} |U_{i,t}^{-}|. && (4.80) \end{aligned}$$

Similar to the discussion on the 3 receivers scenario, we assume an optimal solution $\{\mathcal{U}^{+}, \mathcal{U}^{-}\} = \{S_{1}^{+}, U_{2,t}^{+}, ..., U_{Q,t}^{+}, U_{1,t}^{-}, U_{2,t}^{-}, ..., U_{R,t}^{-}\}$ for which the delivery rate is

$$R_{on}^{t+1*} = P_{1,t+1}(1 - S_{1}^{+}) + \textstyle\sum_{i=2}^{L} P_{i,t+1}(1 - U_{i,t}), \quad \forall t = 1, 2, 3, ..., K-1, \tag{4.81}$$

For any other point that has $U_{1,t}^{+} = S_{1,t}^{+} - \epsilon$, the delivery rate would be

$$\begin{aligned} R_{on}^{t+1,\epsilon} &= P_{1,t+1}(1 - S_{1}^{+} + \epsilon) + \sum_{i=2}^{L} P_{i,t+1}(1 - U_{i,t} - \epsilon_i), && \forall t = 1, 2, ..., K-1, \\ &= P_{1,t+1}(1 - S_{1}^{+}) + \sum_{i=2}^{L} P_{i,t+1}(1 - U_{i,t}) + P_{1,t+1}\epsilon - \sum_{i=2}^{L} P_{i,t+1}\epsilon_i, && \forall t = 1, 2, 3, ..., K-1, \\ &= R_{on}^{t+1*} + P_{1,t+1}\epsilon - \sum_{i=2}^{L} P_{i,t+1}\epsilon_i, \; \forall t = 1, 2, 3, ..., K-1, && (4.82) \end{aligned}$$

where $\sum_{i=2}^{K} \epsilon_i = \epsilon$.Since $P_{2,t+1} > P_{3,t+1} > ... > P_{t+1,K}$, $P_{1,t+1}\epsilon > \sum_{i=2}^{L} P_{i,t+1}\epsilon_i$ and $R_{on,\epsilon}^{t+1}$ is always larger than R_{on}^{t+1*}, and the solution is sub-optimal. The concurrent decrease in the update $U_{1,t}$ by ϵ with an increase of the other $\{U_{i,t}, i = 2, ..., L\}$ by ϵ in total will have two contradicting effects on the expected delivery rate. First, an increase in the expected delivery rate of F_1 and a decrease in the expected delivery rate generated by all other files. However, the decrease in the delivery rate due to the increase in $\{U_{i,t}, i = 2, ..., L\}$ is $\sum_{i=2}^{L} P_{i,t+1}\epsilon_i$, which is smaller than the increase in the delivery rate of F_1 ($P_{1,t+1}\epsilon$). Accordingly, the net change in the delivery rate due to a decrease in $U_{1,t}$ by ϵ is always positive and its value is lower-bounded by $(P_{1,t+1} - P_{2,t+1})\epsilon$. As such, any point that has $U_1^+ \leq S_{1,t}^+$ is sub-optimal. Using the same argument for all $U_{i,t}^+, i = 1, 2, ..., Q$, and $U_{i,t}^-, i = 1, 2, ..., R$, one gets

$$U_{i,t}^+ \leq \min(\min(\eta, \beta_{R+1}) - \alpha_i, S_{i,t}^+) \quad \forall i = 1, 2, ..., Q, \tag{4.83}$$

$$U_{i,t}^- \leq \min(\eta - \beta_i, S_{i,t}^-) \quad \forall i = 1, 2, ..., R, \tag{4.84}$$

$$\alpha_i = \sum_{j=1}^{i-1} U_{j,t}^+, \ \ \forall i = 1, 2, ..., Q, \ \ \beta_i = \sum_{j=1}^{i-1} U_{j,t}^-, \ \ \forall i = 1, 2, ..., R. \tag{4.85}$$

Similar to the three receivers case, the solution of the above equations can be found using an extension of the mentioned algorithms, where we update the highest popularity files first by increasing the sizes of the cached parts then update the lowest popularity files by decreasing their cached sizes. A detailed version of the algorithm is presented below and its complexity analysis follows.

Algorithm 2: LDC procedure

```
for $t = 1 : K$ do
    Initialize the maximum total cache update:
    CacheIncrease $= \min(S_{d_t,t}, 1 - S_{d_t,t})$;
    CacheDecrease = CacheIncrease;
    Set $\mathcal{P}^{t+1} = [P_{1,t+1} ... P_{L,t+1}]$
    while CacheIncrease $\geq 0$ do
        Find File-to-update, $F_l : l = \arg\max(\mathcal{P}^{t+1})$;
        Set: $S_{i,t+1} = \min(2S_{l,t}, S_{l,t} + \text{CacheIncrease})$
        Update: CacheIncrease $\leftarrow$ CacheIncrease- $(S_{l,t+1} - S_{l,t})$
        $\mathcal{P}^{t+1} \leftarrow \mathcal{P}^{t+1}\ l$
    end
    while CacheDecrease $\geq 0$ do
        Find File-to-update: $F_l : l = \arg\min_i(\mathcal{P}, \rangle^{t+1})$;
        Set: $S_{i,t+1} = \max(S_{i,t}, \text{CacheDecrease})$
        Update: CacheDecrease $\leftarrow$ CacheDecrease - ( $S_{i,t}$)
        $\mathcal{P}^{t+1} \leftarrow \mathcal{P}^{t+1} \backslash l$
    end
end
```

□

4.6.2 Complexity analysis

We have provided two sub-schemes of our caching scheme. The first scheme solves K optimization problems (4.23), each of dimension K, and each can be solved using interior point methods in $O((m+n)^{1.5}n^2L)$ processing time. Thus, the total processing time is $O((m+n)^{1.5}n^2LK)$, $m = 2^{K+1} + K$ and $d_{\max} = O(K)$. Whereas, the LDC algorithm solves the K online optimization problems in $O(K)$ processing time each for a total of $O(K^2)$ processing time. This can be explained as follows. Investigating the LDC procedure, it is composed mainly of two non-nested while loops at each time slot. The first while loop is for the Q positive cache size updates and the second is for the R negative updates. The subroutine in each loop updates the cache size of $Q, R \leq K - 1$ files. As such, the maximum number of repetitions for both loops combined is $Q + R = K$ times or the total number of files. Each repetition of the subroutines needs $O(1)$ processing time, yielding a total of $O(K)$ processing time for each time slot.

4.7 Results

In this section, we evaluate the performance of the developed scheme and show the advantages of using it in comparison to using other caching schemes available in literature. System simulations is used to evaluate the expected delivery sum rate and expected Age of information. The expected delivery rate of the scheme is compared to that of three different schemes. The uncoded offline caching scheme (UOC) [9], the (LRU) [58] , and the offline coded caching and updates (OCC) (chapter 3). The UOC caches the most popular files once at the placement phase and does not update the caches thereafter. The LRU caches the most recently file sent(used) in the network. This scheme is beneficial if a small group of files sustains having the highest popularities over large periods. However, it performs poorly through rapid variations in the files popularities. The OCC assumes full knowledge of the changes of the file popularities and optimizes all the cache and updates decisions at offline at the placement phase.

The simulations are performed for two types of variations in file popularities. For each type a different sample of the file popularity distribution is used. The first type is file popularity variations with high randomness, the second type assumes that the variations in files popularities are following a trend (i.e., some files have fading popularities and other have growing popularities). The request probabilities of the first and the second types are generated according to

$$P_{i,t} = \frac{\phi_{F_{i,t}}}{\sum_i \phi_{F_{i,t}}}, P_{i,t} = \frac{i^t \phi_{F_{i,t}}}{\sum_{i=1}^{K} i^t} \quad \forall t = 1, 2, ..., K, \ \forall \ i = 1, 2, ..., L, \tag{4.86}$$

respectively, where $\phi_{F_{i,t}} \sim U(0,1); \forall t = 1, 2, 3, ..., K,$ where $\frac{i^t}{\sum_{i=1}^{K} i^t}$ represents the growth/decline trend-line, and $\phi_{F_{i,t}}$ captures the random (relatively smaller) change around the dominant trend-line.

Fig. (4.9) and Fig. (4.10) compare the expected sum delivery sum rate of dynamic coded caching, and LDC to the uncoded caching, LRU, and offline coded caching schemes for the first and the second request models, respectively. The network used for simulations consists of a server with NL bit cache and a number of receiver with N bit cache each (can hold up to one file). The figures show the coded caching gain (reduction of the delivery phase transmission rate) of dynamic coded caching when compared to using uncoded proactive caching or LRU. The figures show the predicted performance degradation compared to the offline coded caching due to the lack of prior full knowledge at the server at the placement phase. However, the dynamic coded caching achieves large reduction in the delivery rate with a small fraction of

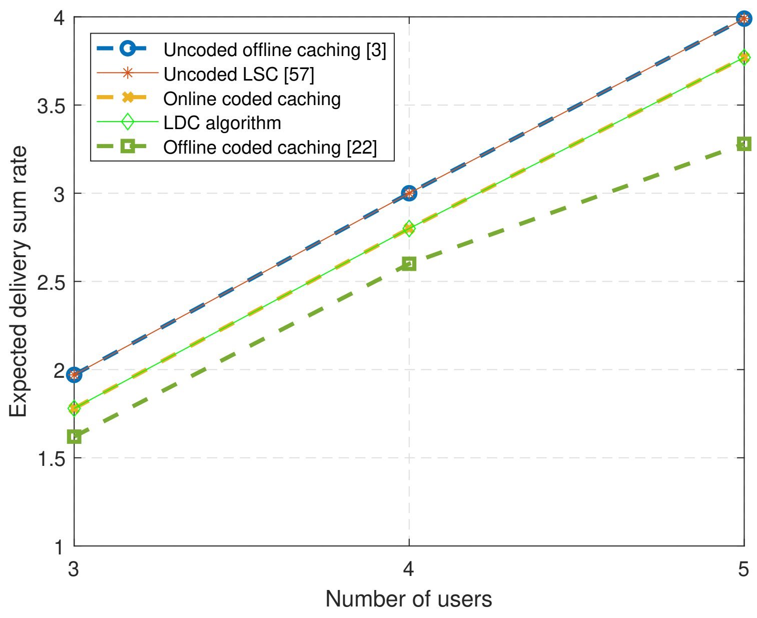


Figure 4.9: The performance of caching schemes compared to other schemes in terms of the delivery rate for different number of receivers for the first request model.

knowledge which shows a *diminishing return* of obtaining knowledge about future file popularities distant in time. Given that the scheme depends only on one time slot knowledge ahead, the scheme is versatile in reacting to unpredictable change in the network rapidly and minimizes the expected delivery sum rate. Compared to offline coded caching that uses full knowledge and to uncoded caching.

Moreover, both figures show that the coding gain grows with the number of receivers in the network. This can be explained by the fact that the aggregate cache of the receivers increases, which in turn offloads more subfiles to the receiver side. It is important to highlight the observed fact that the performance of the proposed scheme is better for the second request model where the file popularities slowly changes. This is due to the fact that the update messages size is limited and the cache updates

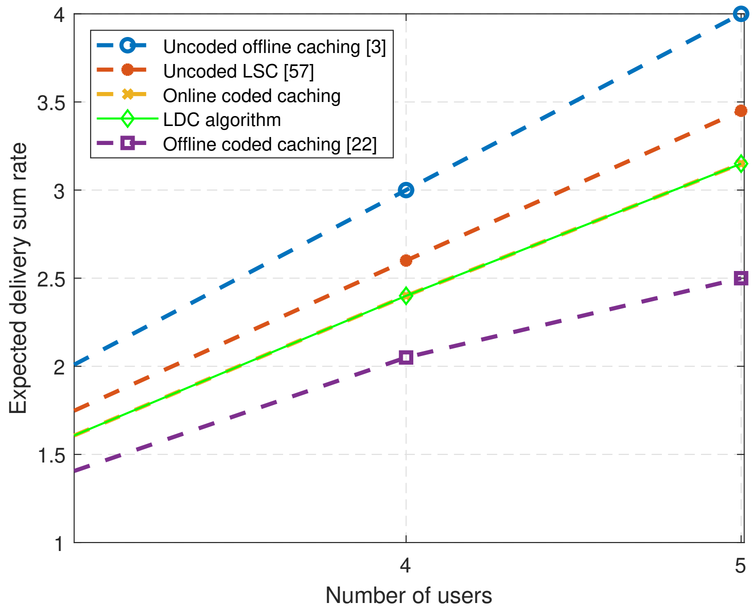


Figure 4.10: The performance of caching schemes compared to other schemes in terms of the delivery rate for different number of receivers for the second request model.

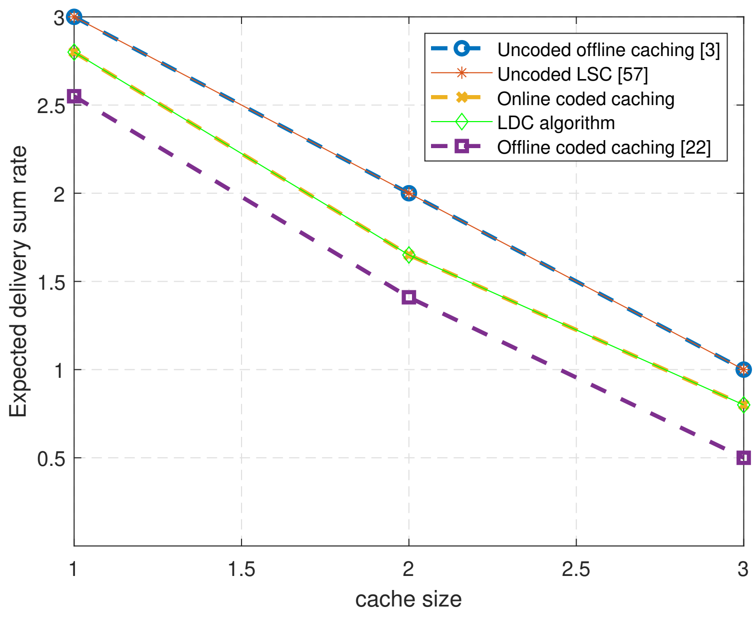


Figure 4.11: The performance of caching schemes compared to other schemes in terms of the expected delivery sum rate with the change in cache size for the first request model.

are limited as well, preventing it from making large changes in the file popularities. Note that in a practical setting, the popularities variations follows the trend-line case (second model) more often. Figures (4.11) and (4.12) show the effect of increasing the cache size on the expected delivery sum rate for a four receivers network with the first and the second request models, respectively. The simulations show that the dynamic coded scheme is more beneficial when the cache size is small due to the optimization it provides for the usage of the limited cache size.

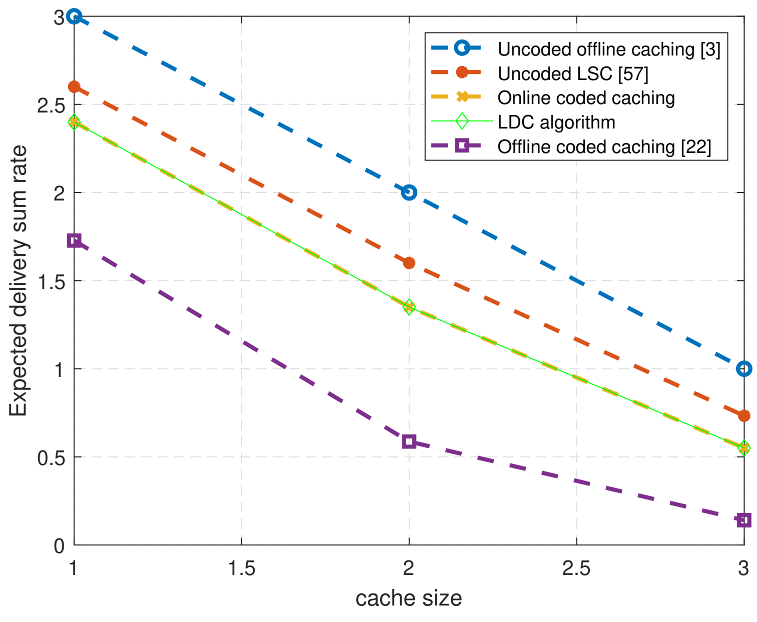


Figure 4.12: The performance of caching schemes compared to other schemes in terms of the expected delivery sum rate with the change in cache size for the second request model.

4.8 Conclusion

We studied a network that consists of a server that has a library of files, and K receivers with future requests limited to the server's library. We proposed a dynamic coded caching scheme for multiple time-varying files features that updates the receiver caches based on the availability of information about the files features. The proposed scheme index-codes the updates with the delivery messages to minimize the expected delivery rate, AoI, among other metrics. The framework is generalized to any set of features. A new non-iterative dynamic caching algorithm with reduced complexity and linear time computation was proposed and its optimality was proved. The algorithm can solve a class of linear problems beyond the one studied in this chapter while keeping the linear time computation. Our numerical results show that our schemes significantly reduce the expected delivery sum rate for all cache sizes and different number of receivers in the system for different distributions/behavior of the files popularities. We showed that proactively and constantly updating the receiver caches reduces the peak (delivery phase) rate of the network even when limited knowledge about the file popularities is available.

Chapter 5

Learning-Based Proactive Coded Caching with Changing File Popularities

Abstract

Proactive caching has risen recently as a promising technique to minimize peak traffic rates in wireless networks with limited resources. This is achieved by proactively storing popular data at different nodes in the network. In this chapter, we consider a wireless network that consists of a base station (BS), with a library of files with changing file popularities, and mobile units MUs that request files from the BS library. We propose a reinforcement learning-based coded caching scheme that constantly updates the MUs caches to match the changing file popularity. The scheme uses index coding to encode the updates with the delivery messages of the requested files by other MUs at no increase in the file delivery rate. To further minimize the delivery rate, the caching and update sizes are optimized. Numerical results show the benefits of the proposed scheme, over conventional caching schemes, in terms of reduced delivery sum rate.

5.1 Introduction

The usage of machine learning techniques is a natural evolution for the design of efficient proactive caching schemes. Machine learning has the flexibility to react to the changing dynamics of the user demands and the complexity of optimizing the available resources accordingly [8, 38, 47, 64–70]. In [38], the authors studied the optimization of cache content of small cells. They modeled the problem as a multi-armed bandit (MAB) problem where the cache content placement is optimized based on the demand history. Machine learning based algorithms were developed for coded proactive caching in [66–70]. The authors of [66] developed a deep reinforcement learning approach for coded caching and coded multicast scheduling to jointly minimize the average delay and power of ultra dense wireless networks. To overcome the problem of large action space for coded caching with time-varying file popularities, the authors of [67] proposed clustering based long short-term memory approach to develop a learning based caching algorithm.

Coded caching has proved to be complex, where finding an optimal coding scheme is cumbersome, especially for a dynamic environment. Motivated by the previous discussion, we study coded caching in a network with changing file popularities and asynchronous delivery. We develop a reinforcement learning-based design of our index-coded caching and updates to minimize the delivery rate, where an agent is trained to design the caching and updates to achieve this aim. Numerical results are provided to show the merits of the proposed schemes, over conventional caching schemes, in terms of reduced delivery sum rate.

5.2 System Model

We consider a wireless network that consists of a BS that has a cached library of L cached M bit files $\mathcal{F} = \{F_1, F_2, ..., F_L\}$, and K cache enabled MUs whose demands are limited to the contents of BS library. Each MU demands a file delivery in a different time slot, and the BS and MUs are assumed to be equipped with one antenna each and limited size caches. The local caches at the MUs are equally sized and can cache up to C files, or CM bits.

The transmission from the BS to the MUs takes place over two phases. In the first (placement) phase, the MUs caches are filled with parts of the files in the BS library $\mathcal{F}$. In the second (delivery) phase, each MU demands one file from $\mathcal{F}$ at a separate time $t \in \{1, 2, ..., K\}$. In particular, the MU demands the missing parts of the file that are not readily available in its cache. Accordingly, the BS sends the demanded missing parts of the file to the respective MU.

The demand-delivery routine of the delivery phase is as follows. one MU demands a file $F_i \in \mathcal{F}$ in a given time slot, then the BS delivers missing parts of that file before the end of the same time slot. We assume that each MU has only one demand per delivery phase (e.g., hour, day). The MUs' demands are not known to the BS a priori. However, the file popularities (demand probabilities), and their changes over time are known to the BS at the placement phase. The probability that a file F_i is demanded at the jth time slot by the jth MU is denoted by $P_{j,i}$. We assume, without loss of generality, that the jth demand belongs to the jth user.

5.2.1 Placement phase

Let $S_{i,1}, \quad i = 1, 2, ..., L$, be the normalized size (with respect to M) of the parts of file F_i that are cached at the placement phase at each MU, i.e., $\sum_{i=1}^{L} S_{i,1} = C$. The MUs have different (possibly overlapping) but equally sized parts of each file.

5.2.2 Delivery phase

At the delivery phase, each user should receive the missing parts of their demanded files. The delivery phase is assumed to consist of K time slots. At the jth time slot, the BS satisfies the demand of the jth MU. The cache content at a given MU can change throughout the delivery phase if additional parts of the library are available at the MUs. However, sending file parts solely to update the local caches is counted towards the delivery sum rate. The delivery rate is minimized by optimizing the normalized sizes of the cached file parts at all time slots $S_{t,i},\ t = 1, 2, ..., K,\ i = 1, 2, ..., L$. Since the actual demands are not known a priori to the BS, and since the file popularities are known, we minimize the expected delivery sum rate rather than the exact sum rate. Since the BS sends the missing part $(1 - S_{t,i})$, if file i is demanded at time t, and since the probability that file i is demanded at time slot t is $P_{i,t}$, the expected delivery sum rate of the delivery phase is expressed as

$$R_d = \sum_{t=1}^{K} \sum_{i=1}^{L} P_{i,t}(1 - S_{t,i}). \tag{5.1}$$

Our objective is to design the information transfer through the placement and delivery phases to minimize the expected delivery sum rate of the delivery phase.

5.3 K Users Learning-Based Coded Caching

Reinforcement learning is a branch of machine learning that studies how an agent learns to take actions that generate the largest predefined cumulative reward by interacting with the environment. In terms of guidance, reinforcement learning differs from supervised learning as it does not need guidance from an experienced learner like providing labeled data. However, it does not learn completely in the dark like unsupervised learning. Instead, it uses the environment's feedback to learn and improve its actions. In this learning process, reinforcement learning has to find a balance between exploration to gain knowledge about the environment and exploitation of current knowledge.

Generally, reinforcement learning can be divided into model free and model-based reinforcement learning [71]. In this work, we consider two scenarios. In the first scenario, the demand probabilities are already known through external help, while in the second scenario, the agent has to learn the demand popularities as well. The role of the reinforcement learning algorithm is to optimize its policy given the known environment model. Markov decision process (MDP) [72] is a good model for sequential decision-making tasks. The process can be represented by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the set of time-varying transition probabilities between states, and $\mathcal{C}$ is the set of costs incurred by the agent actions. In our work, the agent takes proactive cache update decision $\boldsymbol{a}(t) \in \mathcal{A}$, when the system is at state $s(t) \in \mathcal{S}$ according to a policy π. Our aim to is to find a cache update policy $\pi \in \Pi$ that minimizes the expected sum delivery rate of the delivery phase, where Π is the set of feasible policies. One key characteristic of our system is that the file popularities change with time. As a result the transition probability from one cache distribution to another cache distribution is time dependent. To decouple this dependency from the state definition, we add a time feature to our state definition as explained below.

5.3.1 Reinforcement learning model

In the placement phase, the BS (agent) splits each file into $K+1$ subfiles, $F_{i,m}, m = 1, 2, ..., K+1$. The first K subfiles have a normalized size $S_{i,0}$ and are stored at the respective MUs, i.e., subfile $F_{i,k}$ is stored at user k, $\forall i = 1, 2, ..., L,\ k = 1, 2, ..., K$. This caches $KS_{i,0}$ of the file at the MUs while leaving a max $(0,\ 1 - KS_{i,0})$ subfile uncached at any MU and is only available at the BS. The last subfile size can be zero depending on the value of $S_{i,0}$ which is chosen to minimize the expected delivery sum rate. For large subfiles sizes ($KS_{i,0} > 1$), the overlapping subfiles contain all the information in the original file and the file is completely cached/distributed among MUs caches, and the size of $F_{i,K+1}$ is zero. For small subfiles ($KS_{i,0} \leq 1$), the file is not completely distributed among MUs caches and the remaining part is only available at the BS.

During the delivery phase, the BS should deliver the missing subfiles to the MUs at the time of demand. When user k demands file i, the BS sends the remaining part in a delivery message

$$\begin{aligned} M_{\text{delivery}} &= F_i \backslash F_{i,k} \\ &= \{F_{i,2} \cup F_{i,3} ... \cup F_{i,G}\} \backslash F_{i,k}, \\ & G = \begin{cases} K, & KS_i \geq 1, \\ K+1, & KS_i < 1. \end{cases} \end{aligned} \tag{5.2}$$

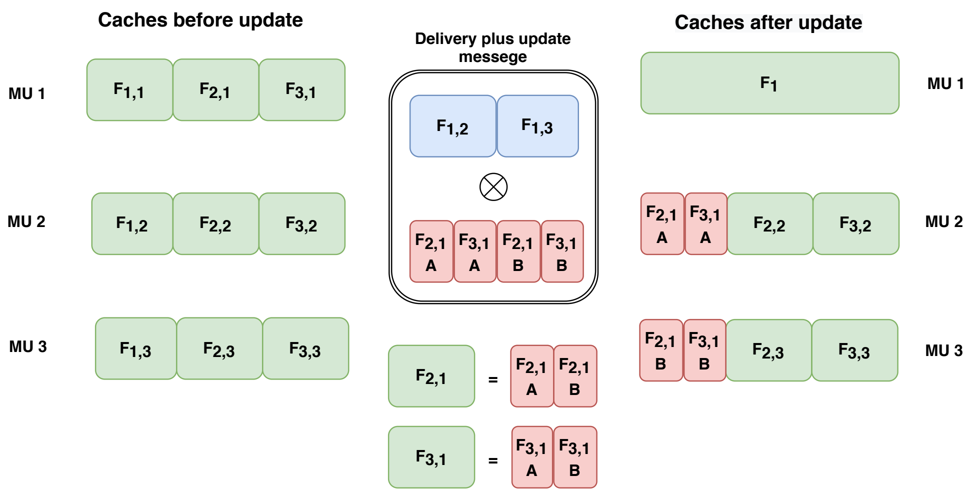


Figure 5.1: Two MUs delivery plus cache update example.

The BS needs to increase the size of the parts belonging to files that have seen popularity increase by sending new parts to the MUs to cache. This has to be at the cost of flushing parts of the files that have seen a popularity decrease. The BS sends the following cache update

$$M_{\text{update}} = \{F_{1,k} \cup ... \cup F_{i-1,k} \cup F_{i+1,k} \cup ... \cup F_{L,k}\}. \tag{5.3}$$

Accordingly, the BS transmits an index coded delivery plus update message

$$M_{\text{joint}} = M_{\text{delivery}} \oplus M_{\text{update}}. \tag{5.4}$$

The system state $s(t)$ at time t is defined by the sizes of the cached file parts at the users $\boldsymbol{S}(t) = [S_{t,1}S_{t,2}...S_{t,L}]$ and the demanded file $d(t) \in \mathcal{F}$ at that instant. The system state can be represented as

$$s(t) = [\boldsymbol{S}(t), d(t), t]. \tag{5.5}$$

Let $a_i(t)$ be the change in the size of the cached part of file i at time t and let $\boldsymbol{a}(t) = [a_1(t), a_2(t), ..., a_L(t)]$ denote the caching action vector at time t, $\boldsymbol{a}(t) \in \mathcal{A}$.

Upon taking an action $\boldsymbol{a}(t)$, the cached file sizes change to

$$\boldsymbol{S}(t+1) = \boldsymbol{S}(t) + \boldsymbol{a}(t). \tag{5.6}$$

Moreover, a new demand $d(t+1)$ arrives at time $t+1$. As a result, the new system state at $t+1$ is

$$s(t+1|\boldsymbol{a}(t)) = [\boldsymbol{S}(t) + \boldsymbol{a}(t), d(t+1), t+1]. \tag{5.7}$$

Since the file demands are probabilistic, the transition probability from state s to another state $\tilde{s}$ given an action $\boldsymbol{a}$, $P_{\boldsymbol{a}}(\tilde{\mathrm{s}}, \mathrm{s})$, is the probability that the sizes of the cached file parts change from $\boldsymbol{S}(\mathrm{s})$ to $\boldsymbol{S}(\tilde{\mathrm{s}})$, while the demand $d(\tilde{\mathrm{s}})$ is subsequent to $d(\mathrm{s})$, i.e.,

$$P_{\mathrm{a}}(\tilde{\mathrm{s}}, \mathrm{s}) = [P_{t,d(\tilde{\mathrm{s}})}|d(t+1) = d(\tilde{\mathrm{s}}), d(t) = d(\mathrm{s}), \boldsymbol{a}(t) = \mathrm{a}]. \tag{5.8}$$

5.3.1.1 Action space

The action space is the possible caching decisions or the changes in cache content at each state. The action space is limited by the delivery and update message size and the ability of each MU to decode the message or parts of it. As explained above, the message sent is an XOR of the missing part sent to the MU making the demand (delivery message) and an update message. However, for any of the MUs to decode the update message, it should have the entire delivery message in its cache. Since each MU cache has a different part of the demanded file (delivery message), it can partially *decode* an equal-sized part of the update message. In the example in Fig. 5.1, there are three MUs in the network, the first MU demands $F_{1,2}$ which is XORed with parts of $F_{2,1}, F_{3,1}$. The second MU can only decode part of $F_{2,1}$. In general terms, the cache *update* of each MU is limited such that the *update* in file F_i at instant j

(i.e., $S_{t+1,i} - S_{t,i}$) is smaller than or equal to $S_{t,d}$ which is the size of the subfile it has in its cache. This can be represented as

$$S_{t+1,i} - S_{t,i} \leq S_{t,d}, \forall\ i = 1, 2, ..., L, \forall t = 1, 2, ..., K - 1. \tag{5.9}$$

For the same reason mentioned above, the sum of any combination of the updates is smaller than the decoded part of the update message, which can be formulated as

$$\begin{gathered}\sum_{F_i \in \mathcal{W}} S_{t+1,i} - S_{t,i} \leq S_{t,d},\ \mathcal{W} = \binom{\mathcal{F}}{l}, \\ \forall\ l = 1, 2, ..., L, \forall\ i = 1, 2, ..., L\ \ \forall\ t = 1, 2, 3, ..., K - 1.\end{gathered} \tag{5.10}$$

Moreover, the cache update of each MU is limited to the size of the whole update message which is in turn limited by the size of the delivery message (the missing part of the demanded file) $1 - S_{t,d}$. This can be formulated as

$$S_{t+1,i} - S_{t,i} \leq 1 - S_{t,d}, \forall i = 1, 2, ..., L,\ t = 1, 2, ..., K - 1.$$

$$\begin{gathered}\sum_{F_i \in \mathcal{W}} S_{t+1,i} - S_{t,i} \leq 1 - S_{t,d},\ \mathcal{W} = \tbinom{\mathcal{F}}{l}, \\ i = 1, 2, ..., L,\ t = 1, 2, ..., K - 1,\ \ \forall\ \ l = 2, 3, ..., K.\end{gathered} \tag{5.11}$$

On the other-hand, at any instant t, the MU demanding file h has no more than $S_{t,i}$ of file F_i. Therefore, the update message cannot contain more than $S_{t,i}$ of $F_i,\ i \neq h$ XORed with the missing part of the demanded file F_h such that the MU demanding F_h can decode the update message and extract the missing part of F_h. As a result, the update size of any subfile of the other files $F_i,\ i \neq h$, at the other MUs' caches can be constrained as

$$S_{t+1,i} - S_{t,i} \leq S_{t,i},\ \forall\ i = 1, 2, ..., L,\ t = 1, 2, ..., K - 1. \tag{5.12}$$

Finally, the file size limitation can be represented as

$$0 < S_{t,i} \leq 1,\ \forall\, i = 1, 2, ..., L,\ \forall\, t = 1, 2, ..., K-1, \tag{5.13}$$

and the MU cache size limitation on the possible cached part sizes at all time slot can be represented as

$$\sum_{i=1}^{L} S_{t,i} = G,\ \forall\, t = 1, 2, ..., K. \tag{5.14}$$

5.3.2 Cost function

The efficiency of the caching scheme is measured in terms of the expected delivery rate. The cost function driving our reinforcement learning is the expected delivery rate at each time slot. If a user demands file i at time t, then the expected delivery rate is equal to $1 - S_{t,i}$. As the demands are probabilistic, the average delivery rate or the average cost at a time t, when the system is at state $s(t)$, is

$$C(s(t), \boldsymbol{a}(t)) = \sum_{i=1}^{L} P_{t,i}(1 - S_{t,i} + \boldsymbol{a}(t)). \tag{5.15}$$

The best policy is the policy that minimizes the average cost over all time slots. Given the afore-defined state space and feasible action set, we define the policy function $\pi : \mathcal{S} \to \boldsymbol{a}$, which maps each state s to the action $\boldsymbol{a}$. In other words, for a system in state $s(t)$, the caching decision (the change in the cached part sizes) is carried out according to an action $a(t) = \pi(s(t))$. Finally, under a policy π the state value function [71] is

$$V_\pi(s(t)) = \lim_{T\to\infty} \mathbf{E} \sum_{\tau=t}^{T} \gamma^{\tau-t} C(s(\tau), \pi(s(\tau))), \tag{5.16}$$

which is the average cost incurred over infinite time, given a discount factor $\gamma \in [0, 1]$ per time slot. Equivalently, after dropping the time index, the Bellman equation [71] provide a recursive form of the state value function

$$V_\pi(s) = C(s, \pi(s)) + \gamma \sum_{\tilde{s} \in \mathcal{S}} P_\pi(s(t))(\tilde{s}, s) V_\pi(\bar{s}). \tag{5.17}$$

5.3.3 Known file popularities

For a known file popularity, solving the bellman equations recursively (dynamic programming) corresponds to solving the main optimization problem. Using dynamic programming is advantages to using linear programming, in our setup, in a sense that the problem is broken to a number of smaller problems which requires less resources (computer memory in practice) in a time. Finding $V_\pi(s)$ is equivalent to solving a system of linear equations for each policy $\pi \in \Pi$. Given that the number of policies is infinite for non-quantized [1] cache parts $\boldsymbol{X}(t), t = 1, 2, ..., K$ and grows exponentially with the quantization factor q, $S_{t,i} \in \{\frac{C}{q}, \frac{2C}{q}, ..., C\}$, it would be computationally prohibitive in practice. As such, a typical approach in reinforcement learning problem is to employ the so-called policy/value iteration algorithms [71]. An optimal policy π^* can be found by solving the Bellman equation using the value iteration algorithm [71].

A value iteration algorithm to solve our problem is presented in Algorithm 5 (RLICCU). The state space is defined according to (5.5), the number of states is controlled by the quanization factor q. The set of possible actions at state s, i.e., $\boldsymbol{A}(s)$ is constrained by the equations (5.9), (5.10), (5.11), (5.11), (5.12), (5.13), and (5.14).

[1] Cache decisions (part size) are chosen from a continuous space, where quantized decision space is limited to a finite number of allowed part sizes.

Algorithm 5 RLICCU caching algorithm

```
for iteration= 1,2,... do
    initialize: $V^*(s) = 0 \; \forall \; s \in \mathcal{S}$
    for $t = 1, 2, ..., K$ do
        for $s = 1, ..., N_S$ do
            $\boldsymbol{a}$= $\underset{\boldsymbol{a}}{\operatorname{argmin}} \; C(s, a) + \gamma P(\tilde{s}|s, \pi(s))V_\pi(\tilde{s})$
        end
        update : $V^*(s) \leftarrow C(s, a) + \gamma \sum P(\tilde{s}|s, a))V_\pi(\tilde{s})$
    end
end
```

5.3.4 Unknown file popularities: Actor-critic

In this scenario, the agent has to learn the file popularites and balance between exploration and exploitation of the learning experience. Actor critic methods are RL temporal difference methods that adopt both policy and value function methods to improve learning accuracy. The actor is responsible for choosing an action based on the current state and the developed policy, while action evaluation through computing the value function is known as the critic. The critic's role is to show the disadvantage of the policy currently being followed by the actor in terms of the long-term reward. The critic output is in the form of a temporal difference error [73], which drives all learning of both actor and critic.

The critic teaches the actor to seek out good states and avoid bad states. The actor and critic participate in a game where by playing, both get to perform better with time and the resulting performance is better compared to them engaging in learning separately.

5.3.4.1 The advantage function

In temporal difference learning, the agents adjust their actions based on the reward/cost prediction error (temporal difference error). In the actor-critic algorithm, the action adjustment is based on the advantage function, which is the difference between the expected cost of following the actor policy and the value function of the current state.

$$H(s,a) = C(s(t),a) + V_\pi(\tilde{s}|\phi) - V_\pi(s|\phi) \tag{5.18}$$

5.3.4.2 Actor policy update

The actor is represented by a neural network with parameters θ, the policy is updated using the policy gradient algorithm [74]. Let $\mathcal{J}(\theta)$ be the reward of a trajectory τ sampled from the policy π,

$$\mathcal{J}_0(\theta) = \mathbf{E}\sum_t C(s(t),\pi(s(t))). \tag{5.19}$$

A low-variance unbiased estimator for the reward gradient using $V_\pi(s)$ as a baseline can be represented as [74]

$$\nabla_\theta \mathcal{J}(\theta) = \sum_i \nabla \log \pi_\theta(a_i|s_i) H_\pi(s_i,a_i), \tag{5.20}$$

where the last equations shows the importance of the advantage function in building a low variance estimator. Then for each iteration the policy parameters θ is a updated as

$$\theta \leftarrow \theta + \nabla_\theta J(\theta). \tag{5.21}$$

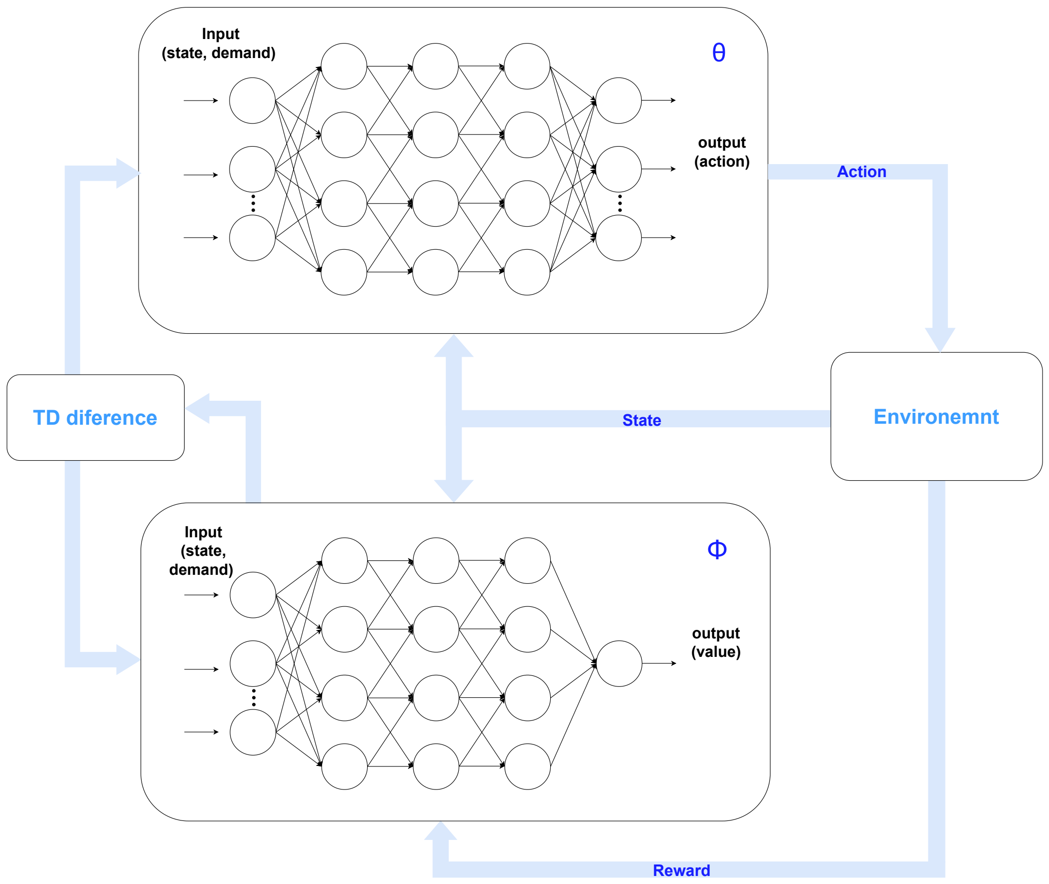


Figure 5.2: Actor-critic neural network with parameters θ and Φ.

5.3.4.3 The critic update

Similarly, the critic is represented by a neural network with parameters Φ to approximate the value function. After each iteration, the parameters are updated to fit the new value function as

$$\theta \leftarrow \Phi + \nabla_{\Phi}\mathcal{L}(\Phi). \tag{5.22}$$

where

$$\mathcal{L}(\Phi) = \frac{1}{2}|V_{\pi}(s) - \sum_{t} C(s_t, \pi(s_t))|. \tag{5.23}$$

Algorithm 6 Actor-critic-caching algorithm

```
for iteration= 1,2,... do
    initialize:
    for $t = 1, 2, ..., K$ do
        H(s,a$_i$) $= C(s, a_i) + \gamma V_{\pi}(\tilde{s}|\phi) - V_{\pi}(s|\phi)$
        $\nabla_{\theta}\mathcal{J}(\theta) = \sum_i \nabla \log \pi_{\theta}(a_i|s_i) H_{\pi}(s_i, a_i)$
        $\theta \to \theta + \alpha \nabla_{\theta} J(\theta)$
        $\nabla_{\theta}\mathcal{L}(\phi) = \sum_i \nabla_{\phi} ||H_{\pi}(s_i, a_i)||$
        $\phi \to \phi + \beta \nabla_{\phi}\mathcal{L}(\phi)$
    end
end
```

5.4 Numerical Results

In this section, we provide numerical results that show the performance of the proposed scheme when compared to the state-of-the-of-art schemes in the literature. Results are shown for a K MUs and L files network. The simulation setup is a network with a cache size of one file. The simulation is performed using the following distribution

$$P_{i,t} = \frac{\phi_{F_{i,t}}}{\sum_i \phi_{F_{i,t}}} \;\; \forall t = 1, 2, ..., K, \; \forall \;\; i = 1, 2, ..., L, \tag{5.24}$$

where $\phi_{F_{i,t}} \sim U(0, 1); \forall t = 1, 2, 3, ...K$. The performance of the developed scheme is compared to uncoded proactive caching [7] (caching the most popular file/s at the placement phase) and last recently used (LRU) caching. LRU is a scheme that continuously updates the local caches by caching the files that were recently sent in the network [75]. The performance metrics used in the comparison are the expected delivery sum rate and the fairness of the resource distribution among MUs. Moreover, the effect of quantization of possible $S_{t,i}, t = 1, 2, ..., K, i = 1, 2, ..., L$ on the performance of the developed RL scheme is discussed.

Fig. 5.3 compares the expected delivery sum rate (in terms of the number of files) of the proposed RLICCU to that of LRU and uncoded proactive caching for a different number of users in the network. The figure shows the caching gain (reduction in delivery phase transmission rate) of the developed RL-based caching update when compared to proactive caching and LRU. The quantization factor q used is equal to the number of files.

Quantization is a key characteristic of RL algorithms that deals with continuous state space. Fig. 5.4 shows the effect of quantization on both times of convergence and

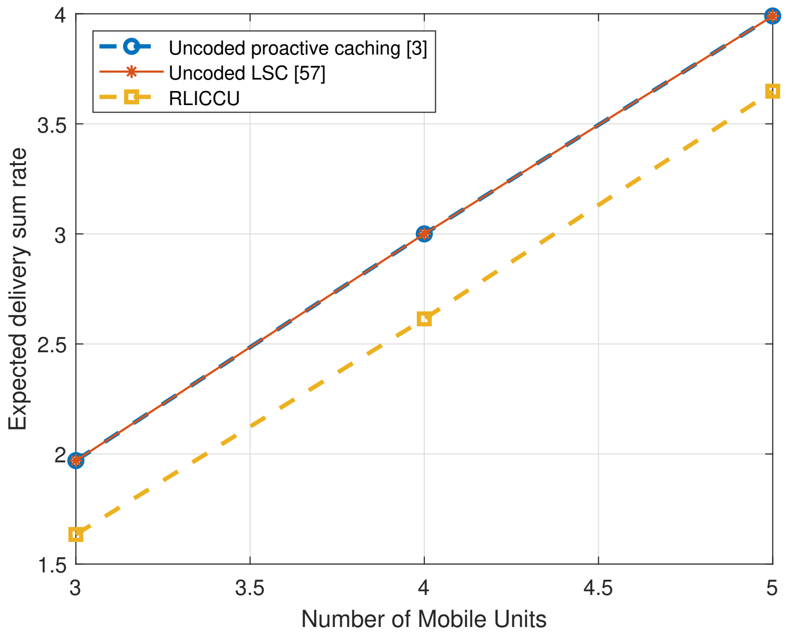


Figure 5.3: Expected delivery sum rate (no. of files) of the RL Caching compared to uncoded caching and LRU.

the expected delivery sum rate for four MUs networks using the RLICCU. The figure shows an almost exponential decrease of the expected sum rate with the increase in the quantization factor, whereas the convergence time grows exponentially with the increase in the quantization factor. From the figure, we can see that a quantization factor that equals eight is a very good choice for a four MUs network resulting in less than 50 seconds till convergence to good set of caching actions for all time slots, whereas a quantization factor of four still achieves a good results in a about 2.5 seconds.

Fairness is an important aspect of wireless network QoS that is rarely considered in proactive caching works. Generally, the minimization of delivery rate and fairness are conflicting goals since caching the most popular files affects the QoS for MUs

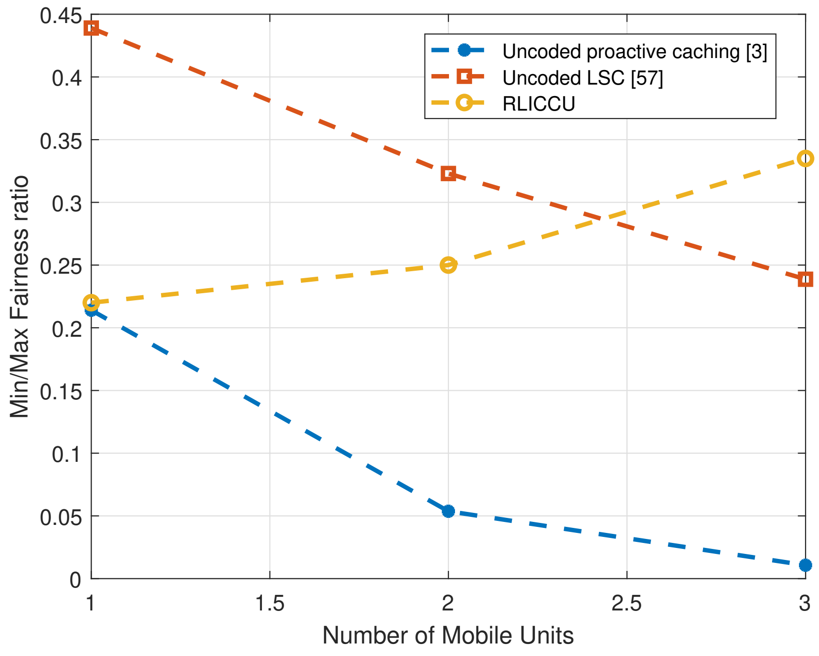


Figure 5.4: Quantization effect on the performance of the RLICCU.

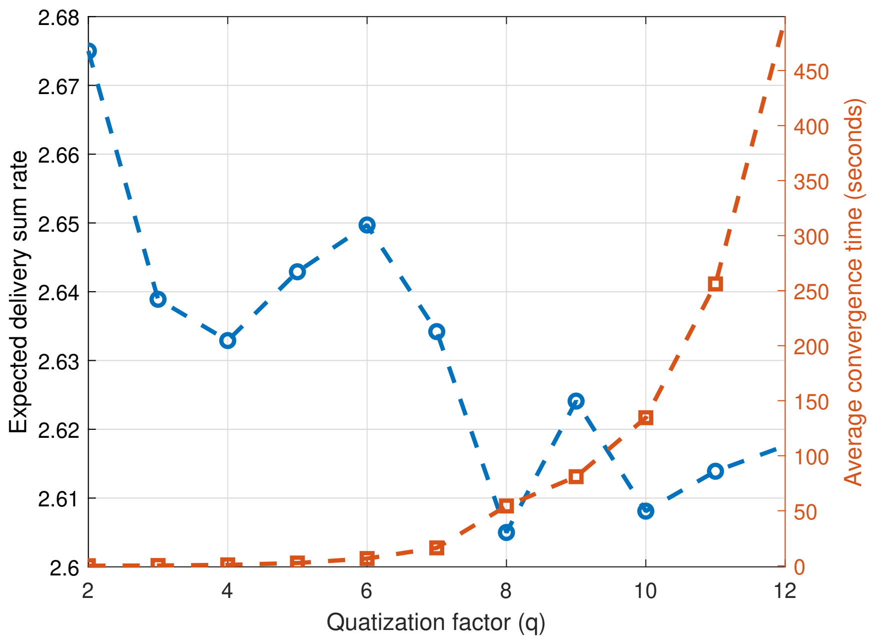


Figure 5.5: Minmax fairness performance of the caching schemes.

demanding the least popular files. Since our scheme caches parts of less popular files at the placement phase, it achieves good performance on the min-max fairness measure (F_{minmax}) compared to other proactive caching algorithms as can be seen from Fig. 5.4. The min-max fairness is defined as

$$F_{\text{minmax}} = \frac{\min(\mathcal{R})}{\max(\mathcal{R})}, \tag{5.25}$$

where $\mathcal{R} = \{R_1, R_2, ..., R_K\}$. Fig. 5.5 shows the min-max fairness performance of the RLICCU scheme compared to uncoded proactive caching and LRU. The RLICCU fairness improves with increasing the number of MUs in the network while the performances of uncoded proactive caching and LRU deteriorate. This is because uncoded proactive caching and LRU focus on a limited number of files and as the number of mobile units increases, the disparity between the rates of the best served MU and worst served MU increases.

5.5 Conclusion

In this chapter, we studied proactive caching for a cellular network with one BS and multiple MUs. We proposed an index coding scheme that constantly updates the MUs' local cache with no increase in the delivery sum rate. We developed a reinforcement learning algorithm that optimizes the cache placement and update independently. We showed that the developed reinforcement learning achieves faster convergence using limited computing resources which presents almost no load to the BS operation.

Chapter 6

Delay Optimization of Cache-Aided Networks with Congested Backhual: a Learning Approach

Abstract

Proactive caching emerged as a great tool to minimize peak traffic rates by storing popular data, in advance, at different nodes in the network. Current studies have mainly focused on proactively storing popular whole files. However, for certain categories of files like videos which constitute the majority of internet traffic, this is not the best strategy delay-wise. As videos can be easily segmented, sending later segments of videos can be less time-critical. In this paper, we study the effect of segmenting video caching decisions under the assumption that the back-haul is congested. We provide a reinforcement learning algorithm for optimizing segmented caching decisions to minimize the average delay. The caching decision is modeled as a combinatorial multi-armed problem and an $O(\log n)$ algorithm is presented for an n number of iterations. The algorithm is then leveraged to gain insights into the caching-queues-delay dynamics. Numerical results show the benefits of the proposed algorithm over conventional caching schemes, in terms of reducing the average delay and improving the fairness of resource distribution.

6.1 Introduction

Cellular network traffic has shifted over the past decade from mainly locally generated instantaneous traffic (voice calls) to centrally generated delay-tolerant bulks of traffic (data-driven communication) [3]. Video constituted 82% of Internet traffic in 2020 [2] and continues to grow. It is expected that proactively cached content will be mainly video. Data traffic and voice traffic differ in their delay tolerance. Data traffic generally experiences a large delay between the time of data generation and the time of demand. In this context, the data can be stored in different nodes along the path between the transmitter and receivers. Proactive caching has attracted much research interest recently as an efficient technique to reduce the peak traffic rate and the delivery sum rate. This is achieved by storing parts of the popular content, at various nodes in the networks, before being demanded by the mobile units (MUs). In a practical sense, proactive caching can minimize the total cost of transmission, as transmitters can optimize the time of performing proactive caching to be in less congested times (e.g overnight) where the resources are in abundance. The authors of [4] proposed the idea of a small-cell network cloud as a means to increase cloud capacity and relieve the backhaul constraints while increasing the peak rate through content caching. In [7], a cache-aided small-cell system is considered where the users are moving through the network and the content demand distribution is assumed to be known. The optimum way of assigning files to various small cells distributed through the network is analyzed, in order to minimize the expected downloading time for files. The extreme majority of proactive caching works is popularity-based caching. For example, the authors of [76] focused on exploiting the spatial and social structure of the network to optimize the caching decisions. To alleviate backhaul congestion, they proposed a proactive caching that caches files during off-peak times based on the file popularity and correlations among users' file request patterns, while leveraging social networks and device-to-device (D2D) communications. In [48], the authors studied the effect of caching on enhancing video streaming quality. They proposed a framework of caching-based video transmission, using layered coding to provide standard definition and high-definition versions of each video. They developed a caching policy that caches each layer of the code according to the popularity of each version.

Proactive caching based on files popularities generally reduces delay as it stores data near end users and bypasses the congested resources of the network links. However, the method of storing the most popular content is not proven to be optimal. Towards that end, other works focused on delay-driven proactive caching [28–34, 77–79]. The authors of [28] developed an algorithm that caches content based on both its popularity and fetching time. They showed that their algorithm is more advantageous in reducing delay than popularity-driven algorithms like least recently used (LRU) algorithm. Segmented caching, where caching is performed segment-wise rather than file-wise, was studied in various works [e.g. [29, 30]]. The authors of [29] provided a web proxy segmented caching algorithm that prioritizes earlier segments of videos. They showed that it is advantageous in terms of the bit-hit ratio and the fraction of requests that require a delayed start. They also showed that segmentation-based caching is especially advantageous when the cache size is limited and files popularities change over time. On the other hand, the authors of [30] focused on per-segment popularity and provided an algorithm that can cache individual file segments based on their popularity and their distance to the file start. In [31], the authors proposed a popularity-aware partial caching algorithm that minimizes the average initial delay of the system while allowing a small deviation from the desired starting point. The authors of [32] focused on cooperative cell caching for mobile networks, where each BS caches popular contents to improve the overall delay performance of users. Likewise, the authors of [33] focused on caching for mobile networks, but with D2D communication. They provided an algorithm that minimizes the average transmission delay for this type of network. The algorithm addresses a more general scenario, in which the values of system parameters potentially change over time. On the other hand, in [34], the authors investigated the trade-off between coded caching gain and delivery delay. They developed a computationally efficient caching scheme that effectively exploits coding opportunities while respecting delivery-delay constraints. The authors in [77] compared the performance of four different video caching algorithms in terms of delay in a RAN with a congested backhaul. They presented a scheduling algorithm for utilizing the backhaul resources to serve the caching requirements while meeting a delay threshold. In [78], the authors proposed a two-tier segment-based D2D caching algorithm to decrease the startup and playback delay of Video on Demand in a cellular network. In their algorithm, the cache of each mobile unit is divided into two cache blocks. The first block is for caching and delivering the earliest section of the most popular video and the second block caches the latter portions fully or partially depending on the users' video-watching behaviour and the popularity of videos. Meanwhile, the authors of [79] proposed a delay-based caching algorithm for D2D networks. They obtained the expression of the average file delivery delay based on the inter-contact user mobility model, which is formulated as an optimization problem. The problem of finding a delay optimal caching algorithm is still open, where the contribution of each file/segment to the overall delay is still unknown.

Caching is typically performed in highly dynamic and complex environments. This has encouraged the usage of machine learning algorithms to learn and adapt to such environments. Using learning algorithms to optimize the proactive caching performance was studied in [8, 35–47]. In [8], content caching is optimized through the use of a reinforcement learning algorithm while the transmitter is oblivious to the request statistics. In [35], the authors proposed an online proactive caching scheme based on a bidirectional deep recurrent neural network (BRNN) model to predict time-series content requests and update edge caching accordingly. While the authors of [36] proposed a novel deep learning-based proactive caching framework in cellular networks. Their deep learning algorithm extracts the features of users and content from small base station traffic data and then uses them to estimate the content popularity at the core network. The caching decisions are then made based on the estimate to obtain higher backhaul offloading and user satisfaction. Despite the described research interest and numerous contributions to delay-based caching.

In this chapter, we focus on delay-driven caching, i.e., We provide an algorithm for proactive segmented caching that is based on delay minimization rather than file/segments popularities. To aid in finding a delay-minimizing algorithm, we study the effect of segmenting video caching decisions under the assumption that the backhaul is congested. We apply the proposed algorithm to a cache-aided cellular network with congested backhauls. In this network, the base stations (BSs) are equipped with limited-sized caches to store content expected to be demanded by the MUs.

The cache placement is designed to minimize the average delay experienced by the MUs. We assume that the files requested by the MUs can each be segmented into a number of segments where each segment is fetched and cached separately. We show the advantage of the strategy of segmentation on the average experience delay. The problem of finding the optimal policy is modeled as a combinatorial multi-armed problem (CMAB). Due to the complex backhaul queue dynamics, finding the optimal solution to the formulated problem is complex. Thus, we provide an approximate solution to the CMAB problem that is asymptotically optimal. Simulation results show the benefits of the proposed scheme over conventional caching schemes based on file popularity, in terms of reducing the delivery average delay.

The remaining of the paper is structured as follows. We present the system model in Section 6.2. The effect of segmentation on the delivery average delay is studied in Section 6.3, while the effect of file length on the delay is studied in Section 6.4.1. We formulate the delay optimization problem and present the segmented caching algorithm in Section 6.6. Simulation results are presented in Section 6.7, and the paper is concluded in section 6.8

6.2 System Model

We consider a wireless network where each BS is connected to the core network through a backhaul link with a download rate R_b. Each BS has a cache of size C that proactively store a number of files. There is a library of files $\mathcal{F} = \{F_i, F_2, ..., F_L\}$, and the users are expected to request one of them at the delivery phase. The files are assumed to have different sizes and each file is segmented into M_i separate segments

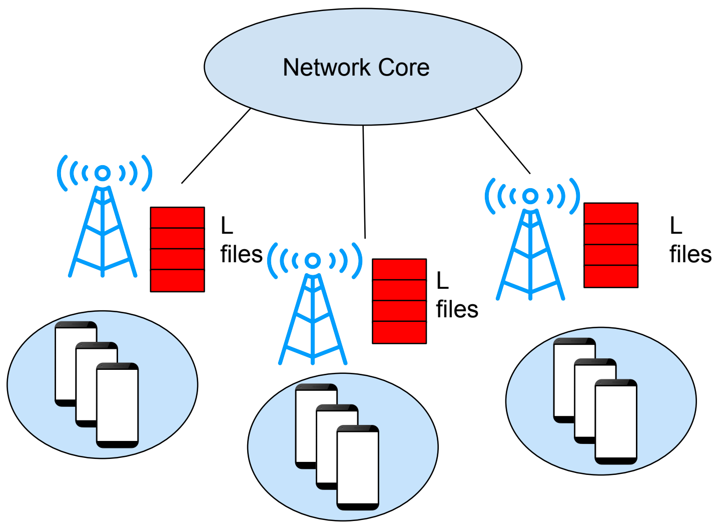


Figure 6.1: System model.

$Z_{i,j}$ of N_s bits each. As such the file library can be described as a library of segments $\mathcal{Z}_L = \{Z_{1,1}, ..., Z_{1,M_1}, ..., Z_{L,M_L}\}$. For simplicity, we assume that the file popularities are identical among users at any time instant. If the requested file is completely available at the BS cache at the time of the user request, it is sent to the requesting user with a channel propagation delay y_{ch}. If some segments are not available at the BS, they are requested from the core network enduring an additional core fetching delay y_{c}. Since the back-haul link is assumed to be congested, file segments requested from the core network will exhibit additional queuing delay y_q. All files are assumed to have the same priority and the back-haul queue has an average service rate μ. On the other hand, the arrival rate λ of the queue would differ according to the caching policy used by the BS. We assume that the caching policy runs through repetitive periods of time (e.g., overnight) denoted by the placement phase. Afterward, through the delivery phase, the users' requests arrive at the BS where they should be satisfied with a minimum delay.

6.2.1 Placement phase

In this phase, the BS proactively fills its cache with parts of the segment library $\mathcal{F}$ according to a predefined caching policy π. The proactive caching is constrained as $\sum_{i=1,j=1}^{M_i,L} I(Z_{i,j})N_s = C$, where $I(Z_{x,y})$ is a function that is equal to one if $Z_{x,y}$ is cached and zero otherwise. The size of the cached parts (i.e., the cached fraction) of file i is denoted S_i. The caching policy is designed such that the segments are chosen to minimize the expected average delay in the network within the delivery phase.

6.2.2 Delivery phase

In the delivery phase, each user should receive the requested segments either directly from the BS cache or the core network through the BS. The probability that a file F_i is requested at time t is denoted $P_{i,t}$. For a scenario where the request probability is constant over the entire delivery phase, the index t is dropped and the request probability is denoted P_i. The delay of each caching policy depends on the cached content in the BSs and the MUs' demands. Since the actual demands and associated delays are not deterministic and not known a priori, we optimize the expected average delay instead.

6.3 Delay Minimization through Segmented Caching

Delay is an important QoS factor, especially for video and music files. Caching files based on their popularity decreases the average sum rate of the delivery phase. However, the delay performance of such an approach is complicated and requires in-depth studies, especially in times of congestion. While caching the most popular files decreases contention for resources and, in turn, decreases the delay for both cached and uncached files, it may not be the best approach delay-wise at times of congestion. We will show that caching parts of unpopular files can significantly decrease the delay for congestion-time requests. This of course comes at the cost of higher delivery rates (back-haul throughput). Since the delay is minimal at times of no congestion, focusing on minimizing the delay at times of congestion is a good strategy for decreasing the average delay for all requests.

6.3.1 Single library-file request

Consider the simple case of a network that receives one file request from the library at a time. For example, assume that the library is composed of two files, A and B, with popularities P_A and P_B, respectively. Assume that both files have the same size $4N_s$, i.e., each file is composed of four segments. Assume that the BS has a cache that can only store one file and that the back-haul link rate is N_s. To elaborate on the effect of segmented caching on the delay experienced by a user that requests one file at a time of congestion, assume that the traffic contention with other file requests results in the requests for $\{A, B\}$ being served by only half of the link capacity $\frac{N_s}{2}$. Thus, one segment can be fetched from the core network every two time-slots or one file can be fetched in eight time-slots. Figs. 6.2 and 6.3 show the delivery of the files and the experienced delay under two policies; caching the most popular file (Fig. 6.2) and caching half of each file (Fig. 6.3). Specifically, it caches two nonadjacent parts of each file including the first part of each file. For the first policy, the cached file A is sent to the user upon request with no back-haul delay. However, if file B was requested, the user would experience a total delay of five time-slots due to congestion (Fig. 6.2). Thus, if the time slot is of length T_s , the average delay for caching the whole file policy, Y^w, for this particular scenario is

$$Y^w = 5P_B T_s. \tag{6.1}$$

Meanwhile, if the second policy is used where half of each file is cached at the BS, the user will experience only one time-slot of delay if he requested either file. In this case, the BS has only to transfer two segments (for example A_2, A_4) from the core network and can instantly send the other two segments (A_1, A_3) cached at the BS to the requesting user. The time needed for the BS to send them to the user and for the user to consume them (A_1, A_3) would compensate for the back-haul delay of transferring (A_2, A_4). The average delay, Y^s, for caching parts $\{A_1, A_3, B_1, B_3\}$, for this scenario is

$$Y^s = T_s. \tag{6.2}$$

As such, partial file caching is more advantageous in terms of delay for $P_B > \frac{1}{5}$. In general, the delay experienced by a user is governed by the difference between the total backhaul time needed for the uncached segment to arrive at the BS and the

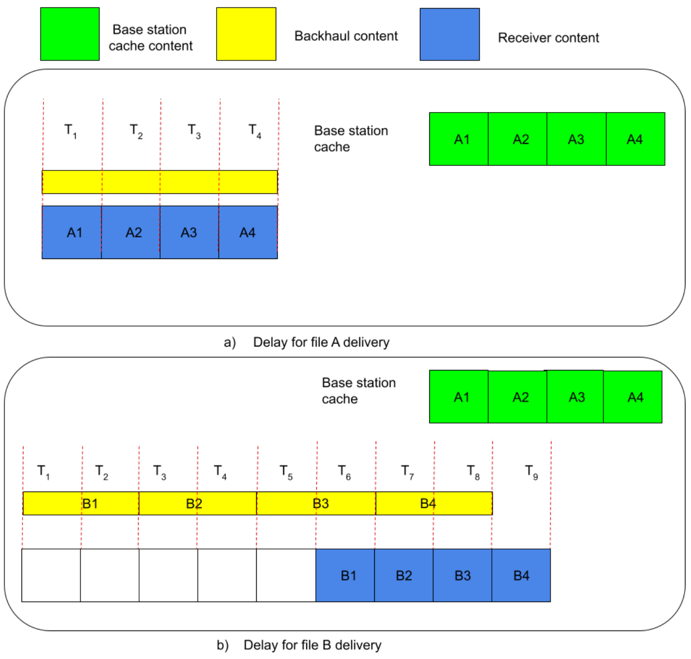


Figure 6.2: Caching most popular files delay with two concurrent requests.

total time for the cached segments to be sent and consumed by the user. For a file of size N_i and an available link capacity R_b, if S_i is the fraction of file i, then the backhaul time T_{BH} can be represented as,

$$T_{\mathrm{BH}} = \frac{(1 - S_i)N_i}{R_b}. \tag{6.3}$$

while the total time for sending and consuming the cached segments is

$$T_{\mathrm{U}} = \frac{N_i}{R_u} - 1. \tag{6.4}$$

, R_u is the rate at which the BS sends and the user consumes the transmitted data. Effectively, R_u is the rate at which the requests for the next segments arrive. Given (6.3) and (6.4), the delay experienced by a user requesting file i is the sum of all segment delays $y_{i,j}^s$, and can be represented as

$$
\begin{aligned}
Y_i^s &= T_{\mathrm{BH}} - T_{\mathrm{U}} + 1, && (6.5)\\
&= \frac{(1-S_i)N_i}{R_b} - (\frac{N_i}{R_u} - 1), && (6.6)\\
&= N_i\left(\frac{(1-S_i)}{R_b} - \frac{1}{R_u}\right) + 1, && (6.7)
\end{aligned}
$$

where (6.9) is due to the fact that the T_{BH} should be at least a one-time slot smaller than T_{U} for the user to experience no delay. Subsequently, the average delay for a network with an L file library is

$$Y^s = 1 + \sum_{i=1}^{L} P_i N_i \left(\frac{(1-S_i)}{R_b} - \frac{1}{R_u}\right). \tag{6.8}$$

The previous average delay can be optimized over the set of cached fractions of the file library $\mathcal{S} = \{S_1, S_2, ..., S_L\}$. Since Y_i^s is non-negative, where a negative value translates to a situation where the segments are being downloaded through the backhaul faster than it is needed, or

$$
\begin{aligned}
Y_i^s &\geq 0, && (6.9)\\
N_i\left(\frac{(1-S_i)}{R_b} - \frac{1}{R_u}\right) + 1 &\geq 0. && (6.10)
\end{aligned}
$$

Equivalently, The cached fractions can then be constrained as

$$S_i \leq 1 - (\frac{R_b(R_u + N_i)}{N_i R_u}), \quad i = 1, 2, ..., L. \tag{6.11}$$

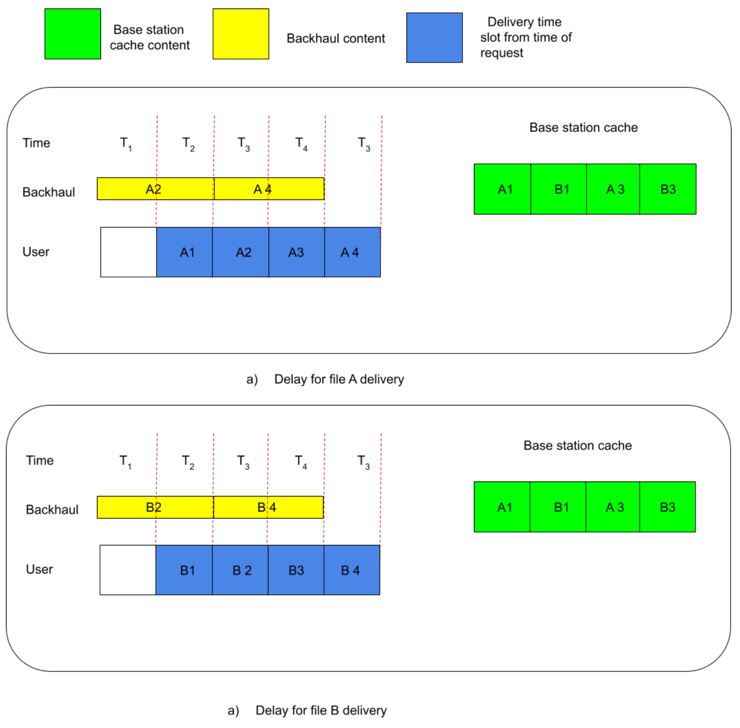


Figure 6.3: Caching half of each file delay with two concurrent requests.

Consequently, the minimum average delay can be represented as

$$\begin{aligned} Y^s = \quad & \min_{S_1,S_2} \quad \sum_{i=1}^{L} P_i Y_i^s, \\ & = 1 + \sum_{i=1}^{L} P_i N_i \left(\frac{(1-S_i)}{R_b} - \frac{1}{R_u} \right) \\ \text{subject to} \quad & 0 < S_i \leq 1 - \left(\frac{R_b(R_u + N_i)}{N_i R_u} \right), \\ & i = 1, 2, ..., L. \end{aligned} \tag{6.12}$$

Meanwhile, the average delay for caching the whole file policy can be represented as

$$Y^w = 1 + \sum_{i \in \mathcal{F}_U} P_i N_i \left(\frac{1}{R_b} - \frac{1}{R_u} \right) \tag{6.13}$$

$$+ \quad P_h N_h \left(\frac{(1-S_h)}{R_b} - \frac{1}{R_u} \right), \tag{6.14}$$

where $\mathcal{F}_U$ is the set of uncached files, h is the index of a file that is partially cached if the size of the cache size in relation to the cached files sizes is not an integer, and S_h is equal to zero if this relation is an integer.

6.3.2 Overlapping library-files requests

Similarly, in the more general case of multiple overlapping file requests, segmented caching focuses on parts that generate more delay rather than parts that are more popular. The benefit of segmented caching is more intuitive in this case. The example shown in Fig. 6.4 consists of three files $\{A, B, C\}$, requested in the same order with two-time slots difference between each request at the begging of time-slots $\{T_1, T_3, T_5\}$, respectively. Assume that file A is the most popular file and that the back-haul link can serve one file at a time. Since the request times are separated by two time-slots each, and due to the overlap of the requested delivery time, the file delivery would result in four time-slots of congestion if no caching is used. Using the most popular file caching policy would relieve two time-slot of congestion (Fig. 6.5) and generate two time-slots of delay. However, using segmented caching can achieve zero delays as shown in (Fig. 6.4). The reason is that the proposed segmented caching method focuses on congestion rather than the system throughput. In general, the delay experienced by file i given the proposed algorithm can be represented as

$$\begin{aligned} Y_i^s \;&=\; Y_{i-1}^s - (T_{r,i} - T_{r,i-1}) + N_i\left(\frac{(1-S_i)}{R_b} - \frac{1}{R_u}\right) + 1, \\ &\forall i = 2, 3, ..., n_r, \end{aligned} \tag{6.15}$$

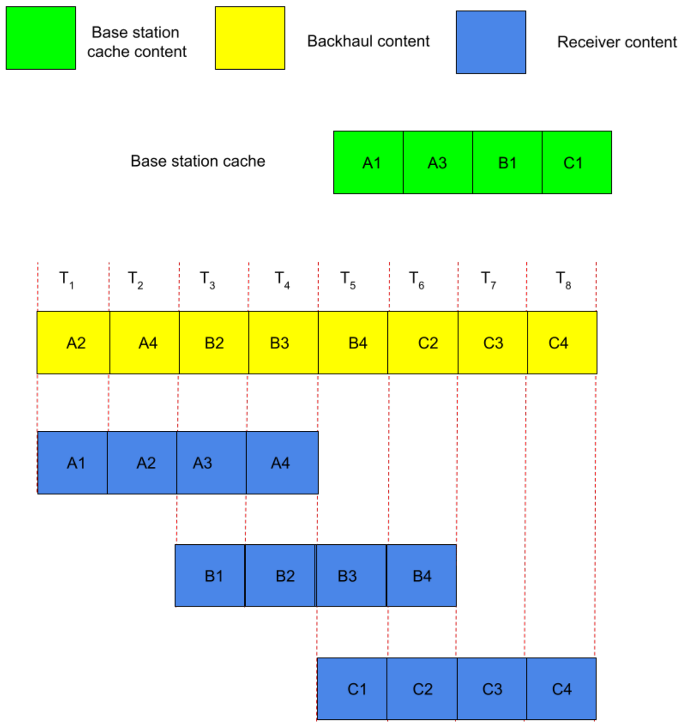


Figure 6.4: Delay for segmented caching for overlapping demands.

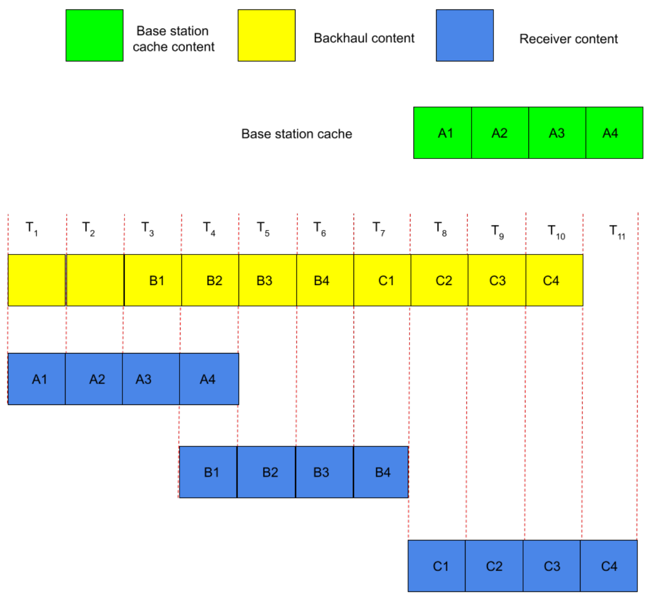


Figure 6.5: Delay for whole file caching for overlapping demands.

where n_r is the number of requested files, $T_{r,i}$ is the time of request of file i, and Y_0 is assumed to be equal to zero due to an empty queue. On the other hand, the overall delay for an overlapping number of requests can be represented as

$$\begin{aligned} Y^s = \quad & \min_{\mathcal{S}} \quad n_r - 2\sum_{i=1}^{n_r}(T_{r,i} - T_{r,i-1}) \\ & +2\sum_{i=1}^{n_r} N_i\left(\frac{(1-S_i)}{R_b} - \frac{1}{R_u}\right) \\ & +N_{n_r}\left(\frac{(1-S_{n_r})}{R_b} - \frac{1}{R_u}\right) \\ \text{subject to} \quad & 0 < S_i \leq 1 - \left(\frac{R_b(R_u+N_i)}{N_i R_u}\right), \\ & i = 1, 2, ..., L. \end{aligned} \tag{6.16}$$

where $\mathcal{S} = \{S_1, S_2, ..., S_L\}$. The previous linear program can be solved using linear optimization methods [56, 80]. The complexity arises from the fact that times $T_{r,i}$

are not deterministic and the fact that the set of possible combinations of demand timings is large. Moreover, the file delivery period is typically composed of successive periods of overlapping and no overlapping demands which compound the system's complexity. In section 6.5, we discuss a reinforcement learning approach to optimize the caching decision given the complexity of the problem.

6.4 Effect of File Features on Delay

This section discusses the effect of other features; the file size and dynamic file popularities, on the choice of the optimal caching segments subset.

6.4.1 The Effect of Video Length (File Size)

The length of the requested files has an important effect on the delays because it affects the queue length of the backhaul link. Small files utilize the backhaul resources in the form of small, possibly overlapping, periods, while large files utilize the backhaul resources in the form of large and less overlapping periods (bursts). While small file requests are more likely to overlap in time and cause congestion, large files' backhaul usage is less dispersed in time and is more likely to contend with (other) requests. Thus, the effect of file size on caching optimality is not straightforward. The choice of the optimal cache set of files/segments is affected by file sizes and the dynamics of requests.

6.4.2 The Effect of dynamic file populartities

Caching typically occurs over the placement phase (e.g overnight) with a little update to the cached content through the delivery (time of congestion). In a practical scenario with a dynamic environment, file popularities are expected to change within the delivery phase [80], where caching decisions are typically made based on the average file popularities. However, in terms of backhaul congestion and the associated generated delay, it is important to focus on the file popularities within the exact congestion periods. Figure 6.6 shows BS traffic data over four consecutive days, where the data was acquired from the open source dataset [81, 82]. The figure shows a spike in traffic once per day. Foremost, the figure shows that the congestion time changes from day to day. The traffic dataset shows a similar pattern for other BSs, where the changes in congestion times range from one to a few hours, while some BSs have multiple congestions per day. Such traffic behavior represents a challenge for popularity based caching. First, in terms of queue dynamics, the caching decisions should be based on the file popularities at times of congestion rather than the average file popularities over the whole delivery period. Second, the prediction error for the time of congestion as well as the popularities within a smaller time frame is expected to be high in practice, which induces a high variability in the algorithm performance. Clearly, a delay-based algorithm is advantageous in such a practical scenario. Moreover, an algorithm that caches multiple parts of different files would have a more stable performance.

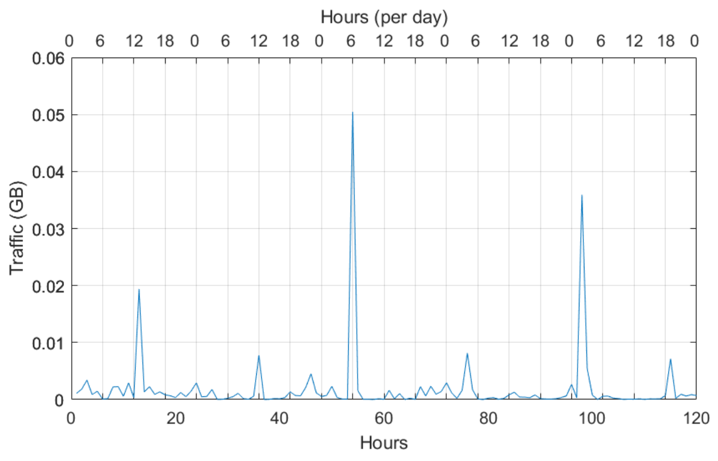


Figure 6.6: Delay for segmented caching for overlapping demands.

6.5 Delay Optimization

The effect of different features of the subset of the cached segments was shown in the previous sections. We instigated the delay experienced when using whole caching and segmented caching policies for a few special cases in terms of the size of the file, requests order, etc. In this section, we formulate the general problem of choosing the optimal subset of segments to be cached. The cached segments are chosen to minimize the average delay of the delivery phase given a limited size cache. Let $\mathbf{Z}_C$ denote the vector of cached segments chosen according to a policy π at the placement phase $t = 0$, and $\mathbf{Z}_U$ denotes the set of the remaining uncached segments of the library $\mathcal{F}$, i.e., $\mathcal{F} = \{\mathbf{Z}_C, \mathbf{Z}_U\}$. Let $y^{\pi}_{i,j,t}$ be the delay experienced by the jth segment of file i at time t when using caching policy π. The network delay at time t is the sum of delays $y^{\pi}_{i,j,t}$ of uncached files and can be expressed as

$$\min_{\mathcal{Z}_C} \quad \sum_{t=0}^{T} \sum_{i,j,Z_{i,j}\in\mathcal{Z}_C} y^{\pi}_{i,j,t}, \tag{6.17}$$

$$\text{subject to} \quad \sum_{i,j,Z_{i,j}\in\mathcal{Z}_C} I(Z_{i,j})N_s = C. \tag{6.18}$$

Note that in contrast to (6.12) and 6.16 has a time index t due to multiple requests over a long period of time. The complexity of the previous problem arises from the coupling of the delays $y^{\pi}_{i,j,t}$ of different segments at different times. This is a result of the fact that the contents of the queue of the congested backhaul depend on the state of the previously demanded segments as well as current demands. As such, the optimization variables $y^{\pi}_{i,j,t}$ are not independent and are a function of caching policy π. While there are approximations for the average delay of similar queues, specifically, burst fed queues (e.g. Poisson Pareto processes) [83], exact delay calculation that does not depend on an exhaustive search for the above case is not known to the best of our knowledge. The problem can be solved exactly through an exhaustive search with exponential complexity. In the next section, we leverage the work of [84] to present a learning-based approach with linear complexity for the cache placement, to minimize the average delay.

6.6 Learning The Cache Placement

The complexity of modeling the cashing-queueing process discussed previously makes the development of a model-free algorithm desirable. In the following, we discuss how reinforcement learning can be a tool for achieving this aim.

6.6.1 The multi-armed bandit problem

The Multi-Armed Bandit (MAB) problem [85] is a resource allocation problem that belongs to decision science (Reinforcement learning). An agent with partial knowledge of the system takes actions that maximize the gain of resources available. At the same time, the agent aspires to acquire new knowledge of the system to improve its future actions, all aiming to maximize long-term gain. The typical MAB model is a machine with A arms. The agent is allowed to pull an arm l at a time and the system generates a reward r_l which is i.i.d over time, with unknown mean. The agent action, a_t, at time t is the decision of which arm to be pulled with the objective to maximize the cumulative reward. A policy π is an algorithm used by the agent to manage its actions (cache content at each placement phase in this paper context). The policy objective is to make the optimal decisions (cache placement) based on the history of the arms pulled (cached segments) and the knowledge of the associated rewards $r(y_i, \mathcal{Z}_C)$ (previous delays) to maximize the expected associated long-term reward (minimize average delay). The expected rewards associated with each arm are estimated based on the observations of the previous rewards. Intuitively, the estimate becomes more accurate as the number of times the arms are pulled increases. Hence, the problem has an associated tradeoff between the exploration of new arms and the exploitation of known arms, a well-known tradeoff in learning problems. If the rewards are known exactly, the optimal policy is simply to pull the arms with higher rewards at each time slot. Accordingly, the *regret* of a policy π can be defined as the loss in the long-term reward compared to the optimal policy.

A MAB problem in which more than one arm is pulled together is a variant known as the CMAB problem [84], where the combination of pulled arms is called a super arm. The problem considered in our work is a typical CMAB problem, due to the coupling of the delays experienced by different segments as explained in the previous section, and the fact that the cache placement can be represented by pulling multiple arms. Solving the CMAB problem is more complicated and more resource-demanding than the traditional MAB problem. If the total number of available arms is X and only x arms can be pulled together at once, the number of possible subsets of arms (superarms) is $\binom{X}{x}$, i.e., the number of subsets of size x that can be formed of the elements of the set of available X arms. As the number of superarms grows exponentially, some works focused on developing CMAB algorithms that are less resource-demanding. The authors of [84] developed an algorithm that uses the information gained at each iteration (superarm pulled) to learn about the ‘‘approximate'' rewards associated with the arms that compose the pulled subset. As such, the learning rate achieved is faster and the achieved regret is $O(\log n)$ for n number of trials.

6.6.2 The combinatorial upper confidence bound algorithm

The combinatorial upper confidence bound (CUCB) algorithm was proposed in [84]. The algorithm has a good theoretical regret and provides a faster learning pace than learning the reward of each arm (file) independently. However, the algorithm does not converge to the optimal solution. The CUCB Algorithm 7 is composed of two main parts; the estimation of the average delay associated with each arm and the action computation. For each superarm played, the agent calculates the average delay vector $\bar{\mathbf{y}} = \{\bar{y}_1, \bar{y}_2, ..., \bar{y}_{A_s}\}$ which is the vectors of the means of the delays experienced by the chosen segments over the past plays. Also, the agent calculates a modified average delay vector $\hat{\mathbf{y}}$ that is a version of the average delay that encourages exploration. The statistics are fed to a computation assistant that calculates the optimal superarm to play next given the statistics.

6.6.3 CUCB Conditions

There are two conditions for the algorithm presented in [84] to be valid for a certain problem. These conditions are monotonicity and bounded smoothness, and they are achieved in our problems since we define the reward as a linear function of the delay. In more detail, we have

- **Monotonicity**: The algorithm requires that the expected reward of playing any super arm $\mathbf{Z} \in \mathcal{Z}$ is monotonically non-decreasing with respect to the expectation vector, i.e., if for all $i, j \in \mathcal{R}, y_i \leq y_j$, we have $r(y_i, Z) \leq r(y_j, Z)$ for all $A \in \mathcal{Z}$.
- **Bounded smoothness**: There exists a strictly increasing invertible function $f(.)$, called bounded smoothness function, such that for any two delay expectation vectors $\bar{y}_i$ and $\bar{y}_j$, we must have $|r(y_i, Z) - r(y_j, Z)| \leq f(\eta)$ if $\max\limits_{i,j} |\bar{y}_i - \bar{y}_j| \leq \eta$.

6.6.4 Computation Assistant (CA)

The algorithm assumes that the agent has a computation assistant that has knowledge of the problem composition, and if given the delay vector $\mathbf{y}$ as the input, it computes the optimal super arm $\mathbf{Z}_C = \arg \max\limits_{\mathbf{Z}_C \in \mathcal{Z}_C} r(\mathbf{Z}_C, \hat{\mathbf{y}})$. Assuming that the average delays associated with segments y_i are known, the average delay can be represented as

$$\min_{\mathcal{Z}_C} \quad \sum_{i,j,Z_{i,j}\in\mathcal{Z}_C} \hat{y}_{i,j}, \tag{6.19}$$

$$\text{subject to} \quad \sum_{i,j,Z_{i,j}\in\mathcal{Z}_C} I(Z_{i,j})N_s = C. \tag{6.20}$$

Compared to (6.17), the previous optimization problem is easier to solve. The reason is that the modified delay estimates $\hat{y}_{i,j}$ are independent. It is a simple linear program that can be solved by choosing $\mathbf{Z}_C$ to include the segments with the highest delay.

Algorithm 7 CUCB based caching

```
1. For each segment l, maintain:
    (a) $T_l$ which represents the number of times the segment $l$ has been cached so far
    (b) $\bar{y}_l$ which is the mean of all outcome delays for $y_l$ for segment $l$ observed so far.
2. Initialize: Play $A_s = \sum_{i=1}^{L} M_i$ rounds, choose the superarms to cache all segments at least once, i.e., for each segment $l$, play an arbitrary super arm $\mathbf{Z}_C \in \mathcal{Z}_C$ such that $z_l \in \mathcal{Z}_C$ and update $T_l$ and $\bar{y}_l$. set $t \leftarrow L$
3. While true do:
    (a) $t \leftarrow t + 1$.
    (b) For each segment $l$, set $\hat{y}_l = \bar{y}_l + \sqrt{\frac{ln3t}{2T_l}}$.
    (c) Calculate the new superarm $Z_C$ = Computation assistant $(\hat{y}_1, \hat{y}_2, ..., \hat{y}_L)$.
    (d) Play $Z_C$.
```

6.7 Results

In this section, we show the advantages of using the proposed segmented caching scheme compared to conventional caching methods used in practice that are based on file popularity. We compare the average delay of the proposed scheme, i.e., caching file segments based on their contribution to system delay, to caching files based solely on their popularity. Also, we exhibit the delay performance of the developed algorithm in terms of delay distribution, and delay fairness between users and compare it to the popularity based caching. Moreover, we investigate the performance of the proposed algorithm to gain insights into the cache-queue-delay dynamics. Finally, we investigate distances to the start of the file of the most cached parts to study the relationship between this distance and the part contribution to delay.

A backhaul link with a capacity of 960 Mbps was considered and all files were assumed to be equal in size (1.2 Gb). Parts of a specific file are equally probable to be requested. The small cell has a cache that can hold one file. The users' demands were generated according to Poisson distribution with a rate of one file per second.

Figure 6.7 shows the total delay experienced by users served by the small cell, over the simulation period, for the two schemes in comparison. The figure shows the reduction of the delay (in seconds) as a result of using the proposed caching scheme compared to caching popular files. The figure shows a growing gap (note that it uses a logarithmic scale on the total delay axis) between using whole file popularity caching and partial delay-based caching. Figure 6.8 compares the performance of both schemes for different backhaul link capacities. The figure shows that the proposed algorithm performs much better than caching popular files in times of congestion

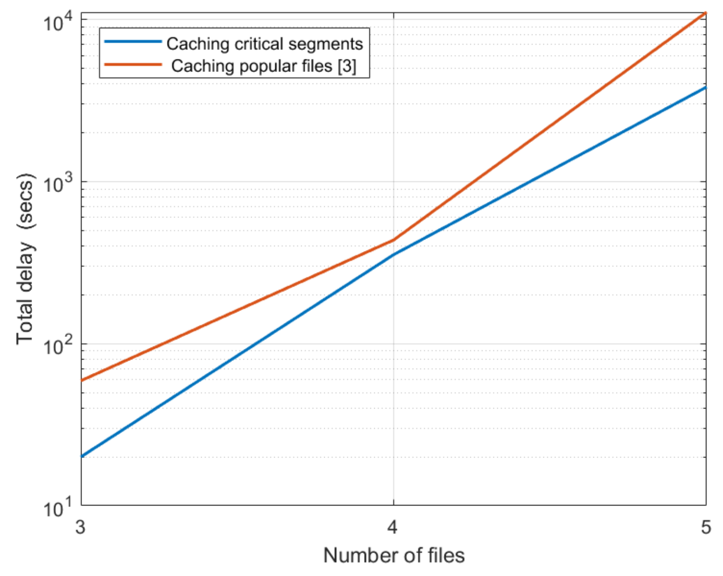


Figure 6.7: Delay vs number of users for whole file caching and segmented caching with overlapping demands.

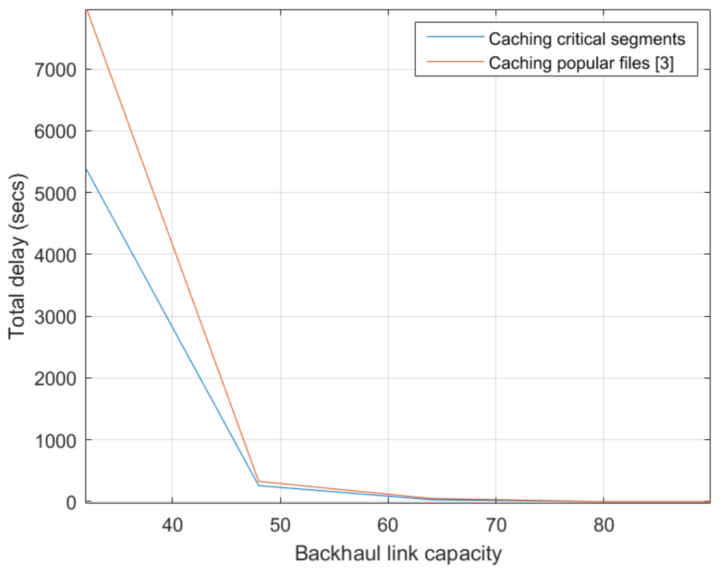


Figure 6.8: The effect of backhaul link capacity on the delay for whole file caching and segmented caching with overlapping demands.

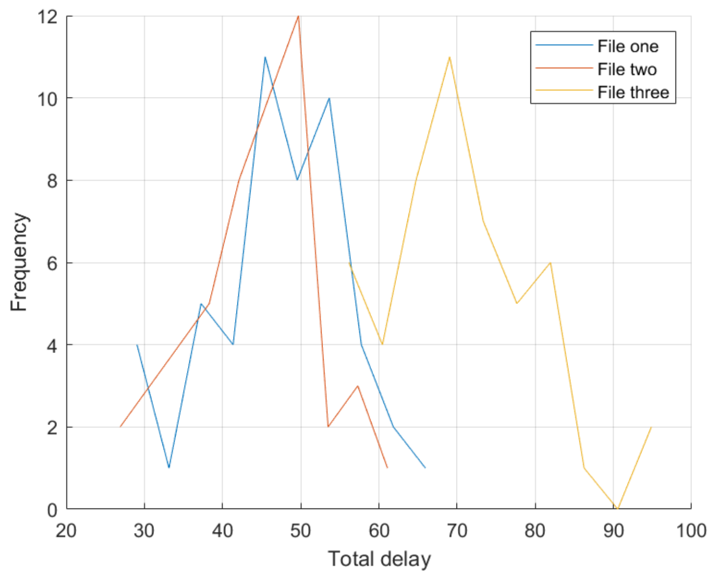


Figure 6.9: Delay Histogram for segmented caching.

(relatively low backhaul link capacity).

Figures 6.9, 6.10 show the delay histograms of the three different files for the segmented caching and whole file caching, respectively. For the whole file caching, a disparity between the delays experienced by users requesting the cached most popular file and the other users, while for the segmented caching the delay histograms for users requesting different files are relatively similar. In a sense, segmented caching provides a more "stable" performance and a more "fair" QoS.

Fairness is an important aspect of wireless networks' QoS that is rarely considered in proactive caching works. Generally, the minimization of delivery rate and fairness are conflicting goals since caching the most popular files affects the QoS for MUs demanding the least popular files. Our scheme achieve good performance on different fairness measures compared to other proactive caching algorithms. The min-max

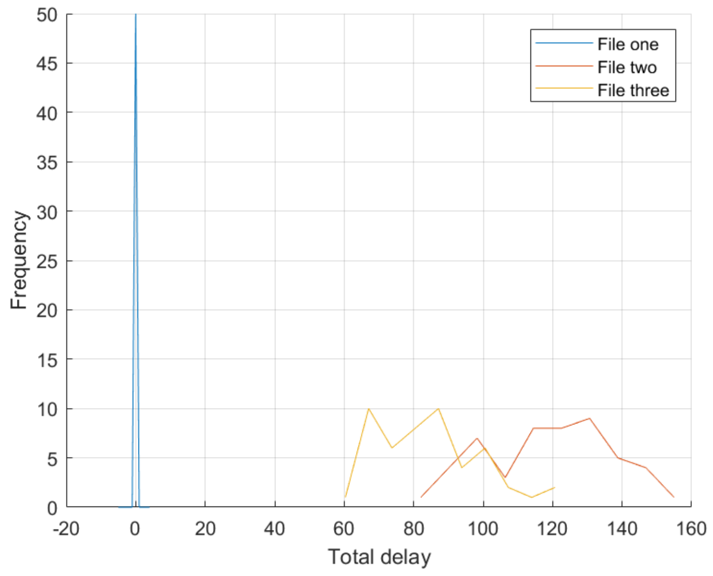


Figure 6.10: Delay histogram for whole file caching.

fairness measure (F_{minmax}) is defined as

$$F_{\text{minmax}} = \frac{\min(\mathcal{D})}{\max(\mathcal{D})}, \tag{6.21}$$

where $\mathcal{D} = \{y_1, y_2, ..., y_L\}$, and $0 \leq F_{\text{minmax}} \leq 1$. The fairest algorithm achieves $F_{\text{minmax}} = 1$. Figure 6.11 shows the min-max fairness performance of the segmented caching scheme compared to the most popular caching scheme. The popularity based caching performs poorly with respect to this measure compared to the developed scheme as expected. The reason is that the most popular files get all the caches and the other files get all the delay.

Figure 6.12 shows the fairness performance according to the Resource Allocation Queuing Fairness Measure (RAQFM) fairness measure [86]. In RAQFM, the underlying principle is that users should be fairly granted an equal share of system resources. While actual delay (or the lack of it) is not an actual system resource like

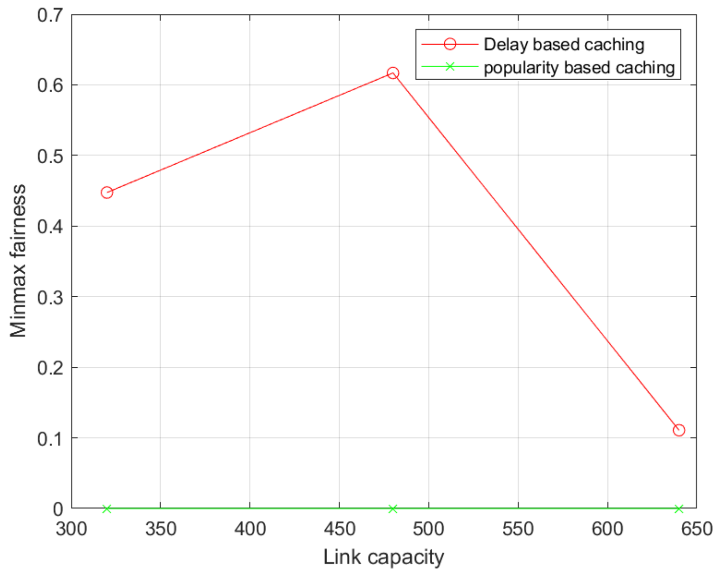


Figure 6.11: Minmax fairness comparison.

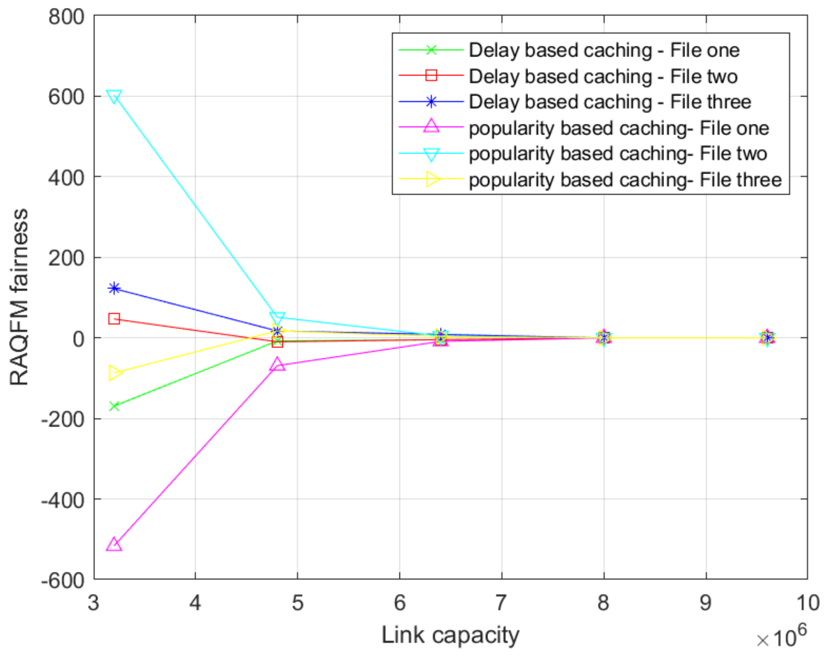


Figure 6.12: RAQFM fairness comparison.

the cache or backhaul rate, the user experience in the system under consideration is mainly governed by delay. To measure the deviation from the fair equally distributed delay reference, the temporal discrimination can be calculated as in [86]

$$TD_i = \bar{y} - y_i, \tag{6.22}$$

where $\bar{y}$ is the average delay in the system. The temporal discrimination measures the amount of deviation away from receiving fair share for a given user, where the fairest algorithm achieves $TD_i = 0$ for all i. Positive temporal discrimination represents an under-served user and negative temporal discrimination represents an over-served user. The figure shows that the developed delay-based algorithm outperforms popularity based caching for all levels of congestion, while their RAQFM fairness performance coincides when the demand is much smaller than the available resources.

Figure 6.13 shows the relative importance of each segment due to its position in the file. It shows the relative normalized average number of times a certain segment has been cached by the developed algorithm. It shows the edge segments are more significant to delay minimization (even if they are less popular) when compared to the middle segments. This result is quite intuitive as earlier segments help with initial delay and relieve congestion with the preceding requested file, while final segments help with queue saturation (self-induced congestion) and congestion with overlapping succeding files. It should be noted that the heat map is affected by different system parameters and this figure shows the average heat map.

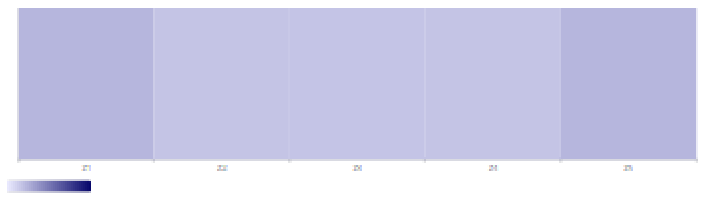


Figure 6.13: Heat map for segments relative importance.

6.8 Conclusion

We studied caching in a cellular network with a congested backhaul. The network is composed of multiple BSs with limited-size caches being leveraged to minimize the average user delay given that the backhauls have large queues due to congestion. We presented a reinforcement learning-based algorithm that optimizes the choice of segments to be cached to minimize the average delay. We showed that caching some segments of less popular files is advantageous delay-wise when compared to popularity-based proactive caching at times of congestion.

Chapter 7

Conclusion and Future Work

In this chapter, we provide insights and conclusions learned through the accomplished work. Moreover, we discuss the directions of future extensions of this work. We studied mobile network where the BS has a library of cached files, and the MUs demands are expected to be limited to that library. We studied different approaches to improve the network QoS through the use of proactive caching.

In Chapters 3, 4, 5, we focused on the case where the file popularities of the library are caching over the delivery period. We studied three different scenarios for popularities information availability at the network. The first is when the information is available offline at the placement phase through external help. The second is when the information is available only one slot ahead. In the third, it is learned by the network overtime. We proved that proactively updating the local finite caches and jointly encoding the files delivery to the MUs over different time slots minimize the delivery sum rate. We developed different algorithms for the three scenarios and evaluated their performance through simulations. In particular, we developed rule-based algorithms for the first and the second scenarios, while we developed a reinforcement learning based algorithm for the third scenario. The complexity of the developed algorithms were discussed. The offline rule-based algorithm depends on an optimization problem that grows exponentially with the number of users requiring large memory resources for large number of users. On the other hand, the dynamic programming offline rule-based algorithm has much less memory requirements at the expense of using more computational resources. The developed reinforcement learning algorithm for the third scenario showed good resources requirements and convergence time.

While not discussed in depth, the algorithms provided in these chapter are rather generic and can be optimized to achieve different targets. AoI, the cache consistency problem, and mobility are direct applications of the provided work. Since the provided algorithms provide zero-rate update to local caches, it can be easily used for cache updates to maintain consistency and/ or respond to user mobility-induced cache update requests.

In Chapter 6, we studied the delay optimization of network with congested backhauls using proactive caching. We showed an interesting, yet counter intuitive, result that caching a chosen set of segments of less popular files is advantageous delay-wise in comparison to popularity based proactive caching. Moreover, we showed that the overall delay is affected by the length of the file whose segment was chosen to be cached. We presented a reinforcement learning based algorithm that optimizes the set segments to be cached in order to minimize the average delay.

Over the course of this work, we came to the conclusion that there are several important problems that need to be studied further, to develop a generic fully functional caching algorithm that performs well in practical wireless networks. We plan to study some of these problems in the near future. In particular, we plan to study the following problems,

- The trade-off between various network parameters like delay, spectrum efficiency, energy,...etc when using coded caching under real wireless networks constraints.
- The design of a generic algorithm that can optimize the coded cache design for a mix of synchronous and asynchronous demands.
- Developing a general upper-bound for coded caching under different practical scenarios.
- Modifying the developed centralize algorithms to include cooperative and D2D caching for further performance improvement.

Bibliography

[1] A. Blasiak, R. Kleinberg, and E. Lubetzky, "Lexicographic products and the power of non-linear network coding," in *Proc. of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Oct. 2011, p. 609–618.

[2] "Cisco visual networking index: Forecast and trends, 2017–2022: White paper." Cisco, Feb. 2019. [Online]. Available: www.cisco.com

[3] "The zettabyte era–trends and analysis." Cisco, July 2016. [Online]. Available: www.cisco.com

[4] E. Baştuğ, J. Guénégo, and M. Debbah, "Cloud storage for small cell networks," in *Proc. IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, Nov. 2012, pp. 100–104.

[5] B. M. K. M. e. a. Baştuğ, E., "Cache-enabled small cell networks: modeling and tradeoffs," *Wireless Com Network 2015*, vol. 41, Aug. 2015.

[6] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation algorithms for mobile data caching in small cell networks," *IEEE Transactions on Communications*, vol. 62, no. 10, pp. 3665–3677, Aug. 2014.

[7] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless video content delivery through distributed caching helpers," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Mar. 2012, pp. 1107–1115.

[8] P. Blasco and D. Gunduz, "Learning-based optimization of cache content in a small cell base station," in *Proc. IEEE International Conference on Communications (ICC)*, June 2014, pp. 1897–1903.

[9] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Transactions on information theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

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

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

[12] A. M. Daniel and W. Yu, "Optimization of heterogeneous coded caching," *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1893–1919, Dec 2020.

[13] R. Pedarsani, M. A. Maddah-Ali, and U. Niesen, "Online coded caching," *IEEE/ACM Transaction on Networking*, vol. 24, no. 2, pp. 836–845, Apr. 2016.

[14] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "The exact rate-memory tradeoff for caching with uncoded prefetching," *IEEE Transactions on Information Theory*, vol. 64, no. 2, pp. 1281–1296, Feb. 2018.

[15] M. M. Amiri, Q. Yang, and D. Gündüz, "Coded caching for a large number of users," in *Proc. IEEE Information Theory Workshop (ITW)*, Sept. 2016, pp. 171–175.

[16] Kai Wan, D. Tuninetti, and P. Piantanida, "Novel delivery schemes for decentralized coded caching in the finite file size regime," in *Proc. IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2017, pp. 1183–1188.

[17] M. A. Maddah-Ali and U. Niesen, "Cache-aided interference channels," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Feb., pp. 809–813.

[18] J. Hachem, U. Niesen, and S. N. Diggavi, "Degrees of freedom of cache-aided wireless interference networks," *IEEE Transactions on Information Theory*, vol. 64, no. 7, pp. 5359–5380, July 2018.

[19] J. Hachem, U. Niesen, and S. Diggavi, "A layered caching architecture for the interference channel," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 415–419.

[20] M. Ji, G. Caire, and A. F. Molisch, "Fundamental limits of caching in wireless D2D networks," *IEEE Transactions on Information Theory*, vol. 62, no. 2, pp. 849–869, Feb. 2016.

[21] Yapar, K. Wan, R. F. Schaefer, and G. Caire, “On the optimality of D2D coded caching with uncoded cache placement and one-shot delivery,” *IEEE Transactions on Communications*, vol. 67, no. 12, pp. 8179–8192, Sept. 2019.

[22] J. Zhang, F. Engelmann, and P. Elia, “Coded caching for reducing CSIT-feedback in wireless communications,” in *Proc. 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sep. 2015, pp. 1099–1105.

[23] S. Jin, Y. Cui, H. Liu, and G. Caire, “Order-optimal decentralized coded caching schemes with good performance in finite file size regime,” in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Dec. 2016, pp. 1–7.

[24] Z. Hu, Z. Zheng, T. Wang, L. Song, and X. Li, “Game theoretic approaches for wireless proactive caching,” *IEEE Communications Magazine*, vol. 54, no. 8, pp. 37–43, Aug. 2016.

[25] H. Ghasemi and A. Ramamoorthy, “Algorithms for asynchronous coded caching,” in *Proc. 51st Asilomar Conference on Signals, Systems, and Computers*, Oct. 2017, pp. 636–640.

[26] Y. Jiang, W. Huang, M. Bennis, and F. Zheng, “Decentralized asynchronous coded caching design and performance analysis in fog radio access networks,” *IEEE Transactions on Mobile Computing*, vol. 19, no. 3, pp. 540–551, Feb. 2020.

[27] M. Amir, E. Bedeer, M. H. Ahmed, and T. Khattab, “Joint coding for proactive caching with changing file popularities,” in *Proc. IEEE 28th*

Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Oct. 2017, pp. 1–6.

[28] P. Scheuermann, J. Shim, and R. Vingralek, "A case for delay-conscious caching of web documents," *Computer Networks and ISDN Systems*, vol. 29, no. 8, pp. 997 – 1005, Sept. 1997.

[29] K.-L. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," in *Proc. of the 10th International Conference on World Wide Web*. New York, NY, USA: Association for Computing Machinery, May 2001, p. 36–44.

[30] Kun-Lung Wu, P. S. Yu, and J. L. Wolf, "Segmentation of multimedia streams for proxy caching," *IEEE Transactions on Multimedia*, vol. 6, no. 5, pp. 770–780, Sept. 2004.

[31] L. Shen, W. Tu, and E. Steinbach, "A flexible starting point based partial caching algorithm for video on demand," in *Proc. 2007 IEEE International Conference on Multimedia and Expo*, July 2007, pp. 76–79.

[32] X. Li, X. Wang, S. Xiao, and V. C. M. Leung, "Delay performance analysis of cooperative cell caching in future mobile networks," in *Proc. 2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 5652–5657.

[33] Y. Wang, X. Tao, X. Zhang, and G. Mao, "Joint caching placement and user association for minimizing user download delay," *IEEE Access*, vol. 4, pp. 8625–8633, Dec. 2016.

[34] U. Niesen and M. A. Maddah-Ali, "Coded caching for delay-sensitive content," *CoRR*, vol. abs/1407.4489, 2014. [Online]. Available: http://arxiv.org/abs/1407.4489

[35] L. Ale, N. Zhang, H. Wu, D. Chen, and T. Han, "Online proactive caching in mobile edge computing using bidirectional deep recurrent neural network," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5520–5530, 2019.

[36] S. Rathore, J. H. Ryu, P. K. Sharma, and J. H. Park, "Deepcachnet: A proactive caching framework based on deep learning in cellular networks," *IEEE Network*, vol. 33, no. 3, pp. 130–138, 2019.

[37] T. Hou, G. Feng, S. Qin, and W. Jiang, "Proactive content caching by exploiting transfer learning for mobile edge computing," *International Journal of Communication Systems*, vol. 31, no. 11, p. e3706, July 2018.

[38] P. Blasco and D. Gündüz, "Learning-based optimization of cache content in a small cell base station," in *Proc. IEEE International Conference on Communications (ICC)*, 2014, pp. 1897–1903.

[39] S. O. Somuyiwa, A. György, and D. Gündüz, "A reinforcement-learning approach to proactive caching in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1331–1344, June 2018.

[40] E. Baştuğ, M. Bennis, E. Zeydan, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, "Big data meets telcos: A proactive caching perspective," *Journal of Communications and Networks*, vol. 17, no. 6, pp. 549–557, 2015.

[41] 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.

[42] A. Sadeghi, G. Wang, and G. B. Giannakis, "Adaptive caching via deep reinforcement learning," *CoRR*, vol. abs/1902.10301, 2019. [Online]. Available: http://arxiv.org/abs/1902.10301

[43] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *Proc. 2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, 2018, pp. 1–6.

[44] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5g using reinforcement learning of space-time popularities," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 180–190, 2018.

[45] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 10 190–10 203, 2018.

[46] W. Jiang, G. Feng, S. Qin, T. S. P. Yum, and G. Cao, "Multi-agent reinforcement learning for efficient content caching in mobile d2d networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1610–1622, 2019.

[47] Y. Zhou, M. Peng, S. Yan, and Y. Sun, "Deep reinforcement learning based coded caching scheme in fog radio access networks," in *Proc. IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, 2018, pp. 309–313.

[48] L. Wu and W. Zhang, “Caching-based scalable video transmission over cellular networks,” *IEEE Commun. Letters*, vol. 20, no. 6, pp. 1156–1159, Apr. 2016.

[49] X. Xu, Y. Zeng, Y. L. Guan, and R. Zhang, “Overcoming endurance issue: Uav-enabled communications with proactive caching,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1231–1244, 2018.

[50] M. Bayat, C. Yapar, and G. Caire, “Spatially scalable lossy coded caching,” in *Proc. 15th International Symposium on Wireless Communication Systems (ISWCS)*, Aug. 2018, pp. 1–6.

[51] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, “Edge-facilitated wireless distributed computing,” in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Dec. 2016, pp. 1–7.

[52] A. Tang, S. Roy, and X. Wang, “Coded caching for wireless backhaul networks with unequal link rates,” *IEEE Transactions on Communications*, vol. 66, no. 1, pp. 1–13, Jan. 2018.

[53] P. Hassanzadeh, A. M. Tulino, J. Llorca, and E. Erkip, “On coding for cache-aided delivery of dynamic correlated content,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1666–1681, Aug. 2018.

[54] C. Zhong, M. C. Gursoy, and S. Velipasalar, “A deep reinforcement learning-based framework for content caching,” in *Proc. 2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, Mar. 2018, pp. 1–6.

[55] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu, "Understanding performance of edge content caching for mobile video streaming," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1076–1089, May 2017.

[56] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, Mar. 2004.

[57] J. Renegar, "A polynomial-time algorithm, based on newton's method, for linear programming," *Mathematical Programming*, vol. 40-40, no. 1-3, pp. 59–93, Jan. 1988.

[58] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management replacement algorithm," in *Proc. the 20th International Conference on Very Large Data Bases*, ser. VLDB '94, Sept., pp. 439–450.

[59] M. Amir, E. Bedeer, M. H. Ahmed, and T. Khattab, "Joint coding for proactive caching with changing file popularities," in *Proc. IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Oct. 2017, pp. 1–6.

[60] ——, "Coded caching for time-varying files popularities and asynchronous delivery," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1458–1472, Jun. 2021.

[61] H. Tang, P. Ciblat, J. Wang, M. Wigger, and R. Yates, "Age of information aware cache updating with file- and age-dependent update durations," in *2020 18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*, June.

[62] S. Zhang, J. Li, H. Luo, J. Gao, L. Zhao, and X. S. Shen, "Towards fresh and low-latency content delivery in vehicular networks: An edge caching aspect," in *Proc. 2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*, Oct. 2018, pp. 1–6.

[63] R. J. Vanderbei, *Linear programming foundations and extensions.* Springer, Dec. 1996, vol. 196.

[64] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5G using reinforcement learning of space-time popularities," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 180–190, Feb. 2018.

[65] S. O. Somuyiwa, A. György, and D. Gündüz, "A reinforcement-learning approach to proactive caching in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1331–1344, June 2018.

[66] Z. Zhang, H. Chen, M. Hua, C. Li, Y. Huang, and L. Yang, "Double coded caching in ultra dense networks: Caching and multicast scheduling via deep reinforcement learning," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 1071–1086, Feb. 2020.

[67] Z. Zhang and M. Tao, "Deep learning for wireless coded caching with unknown and time-variant content popularity," *IEEE Transactions on Wireless Communications*, vol. 20, no. 2, pp. 1152–1163, Feb. 2021.

[68] ——, “Accelerated deep reinforcement learning for wireless coded caching,” in *Proc. 2019 IEEE/CIC International Conference on Communications in China (ICCC)*, Aug. 2019, pp. 249–254.

[69] X. Wu, J. Li, M. Xiao, P. C. Ching, and H. V. Poor, “Multi-agent reinforcement learning for cooperative coded caching via homotopy optimization,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 8, pp. 5258–5272, Aug. 2021.

[70] S. Gao, P. Dong, Z. Pan, and G. Y. Li, “Reinforcement learning based cooperative coded caching under dynamic popularities in ultra-dense networks,” *CoRR*, Mar. 2020. [Online]. Available: http://arxiv.org/abs/2003.03758.

[71] C. Szepesvári, *Algorithms for Reinforcement Learning.* Morgan Claypool, June 2010.

[72] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley Sons, Inc, Aug. 1994.

[73] S. Agrawal, “Policy gradient methods,” May 2018. [Online]. Available: https://ieor8100.github.io/rl/.

[74] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, “A survey of actor-critic reinforcement learning: Standard and natural policy gradients,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, Nov. 2012.

[75] A. Giovanidis and A. Avranas, “Spatial multi-lru caching for wireless networks with coverage overlaps,” in *Proc. of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, ser. SIGMETRICS ’16, Sept. 2016, p. 403–405.

[76] E. Baştuğ, M. Bennis, and M. Debbah, “Social and spatial proactive caching for mobile data offloading,” in *2014 IEEE International Conference on Communications Workshops (ICC)*, 2014, pp. 581–586.

[77] H. Ahlehagh and S. Dey, “Video caching in radio access network: Impact on delay and capacity,” in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2012, pp. 2276–2281.

[78] N. A. et al., “Efficient algorithms for cache-throughput analysis in cellular-D2D 5G networks,” *Computers, Materials & Continua*, vol. 67, no. 2, pp. 1759–1780, Feb. 2021.

[79] R. Sun, Y. Wang, L. Lyu, N. Cheng, S. Zhang, T. Yang, and X. Shen, “Delay-oriented caching strategies in d2d mobile networks,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 8529–8541, Aug. 2020.

[80] M. Amir, E. Bedeer, T. Khattab, and T. M. Ngatched, “Dynamic caching for files with rapidly-varying features and content,” *IEEE Trans. on Commun.*, Oct. 2022.

[81] “City-cellular-traffic-map,” *CoRR*, accessed Sep. 2022 2020. [Online]. Available: https://github.com/caesar0301/city-cellular-traffic-map

[82] S. Q. W. H. K. J. Xiaming Chen, Yaohui Jin, "Analyzing and modeling spatio-temporal dependence of cellular traffic at city scale," in *IEEE International Conference on Commun. (ICC)*, June 2015.

[83] M. Z. Ronald G. Addie, Timothy D. Neame, "Performance evaluation of a queue fed by a poisson pareto burst process," *Computer Networks*, vol. 40, no. 3, pp. 377–397, Oct. 2002.

[84] W. Chen, Y. Wang, and Y. Yuan, "Combinatorial multi-armed bandit: General framework and applications," in *Proc. of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 1. Atlanta, Georgia, USA: PMLR, June 2013, pp. 151–159.

[85] J. C. Gittins, "Bandit processes and dynamic allocation indices," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 41, no. 2, pp. 148–164, Sept. 1979.

[86] D. Raz, H. Levy, and B. Avi-Itzhak, "A resource-allocation queuing fairness measure," vol. 32, June 2004, pp. 130–141.

Appendix A

Index Coding

In this subsection, we define index coding [1] and the underlining transmission problem it solves. The index coding transmission problem is a situation where a set of W independent messages $\mathcal{M} = \{M_1, M_2, ..., M_W\}$ are available at the transmitter, and we need to send different subsets of them to K receivers. The kth receiver requests a set of messages $M_k \subset \mathcal{M}$, while it already obtained another set of message A_k as side information. A receiving node does not need a message that is already available to it, i.e., $M_k \cap A_k = \emptyset$. An index code $\Pi_n(\mathcal{Z}, n, R)$ is used by the transmitter to fulfill the destination needs. The code $\Pi_n(\mathcal{Z}, n, R)$ is composed of a finite alphabet $\mathcal{Z}$ of cardinality $|\mathcal{Z}| > 1$, a joint encoding function, h^c, and a separate decoding function, $h^d_{k,i}$ for the message M_i at the kth receiver. The encoding function h_c maps all the messages to the sequence of transmitted symbols $h^c(M_1, M_2, ..., M_K) = X^n$, where $X^n \in \chi^n$ is the sequence of symbols transmitted over n channel uses. A message $M_k, k \in 1, 2, ..., W$, is a random variable uniformly distributed over the set $F_k \in \{1, 2, ..., |X|\}^{nR_w}$, where $R \in \mathcal{R}^W_+$ is a rate vector $R = (R_1, R_2, ..., R_W)$ in positive real vector space that satisfies the condition that $|X|^{nR_w}$ is an integer for all $1, 2, ..., W$. The decoding function at each receiving node is $g_{k,i}(X_n, A_k) = \hat{M}_{k,i}, \forall i$, where $M_i \in M_k$. An achievable rate tuple $R = (R_1, R_2, ..., R_K) \in R_+^M$ exists if for each $\epsilon, \delta > 0$, there is $(X^n, n, (\bar{R}_1, \bar{R}_2, ..., \bar{R}_K))$ coding scheme, for some X^n, n, such that $\forall w \in 1, 2, ..., W, \bar{R}_w \geq R_w - \delta$.