A MEDIA CACHING APPROACH UTILIZING SOCIAL GROUPS INFORMATION IN 5G EDGE NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Afra Dömeke
March 2021

A MEDIA CACHING APPROACH UTILIZING SOCIAL GROUPS INFORMATION IN 5G EDGE NETWORKS
By Afra Dömeke
March 2021

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

İbrahim Körpeoğlu(Advisor)

Ertan Onur

Özgür Ulusoy

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

A MEDIA CACHING APPROACH UTILIZING SOCIAL GROUPS INFORMATION IN 5G EDGE NETWORKS

Afra Dömeke
M.S. in Computer Engineering
Advisor: İbrahim Körpeoğlu
March 2021

Increased demand for media content by mobile applications has imposed huge pressure on wireless cellular networks to deliver the content efficiently and effectively. To keep up with this demand, mobile edge computing (MEC), also called multi-access edge computing, is introduced to bring cloud computing and storage capabilities to the edges of the cellular networks, such as 5G, with the aim of increasing quality of service to applications and reducing network traffic load. One important application of multi-access edge computing is data caching. As significant portion of multimedia data traffic is generated from media sharing and social network services, various mobile edge caching schemes have emerged to improve the latency performance of these applications. In this thesis, driven from the fact that social interaction between mobile users has a strong influence on data delivery patterns in the network, we propose a socially-aware edge caching system model and methods that consider social groups of users in caching decisions together with storage and transmission capacities of edge servers. Unlike other studies, where users are manually grouped according to their interests, our approach is based on user-specified social groups, where users in a group are neither obligated to share the same interests nor be attentive to the shared content. Our methods cache content considering locations of members of social groups and the willingness of these members in using the related applications. We evaluate the performance of our proposed methods with extensive simulation experiments. The results show that our methods can significantly reduce user-experienced latency and network load.

Keywords: 5G networks, multi-access edge computing.

ÖZET

5G AĞLARINDA SOSYAL GRUP BİLGİLERİNE DAYALI VERİ ÖNBELLEKLEME YÖNTEMİ

Afra Dömeke
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: İbrahim Körpeoğlu
Mart 2021

Günümüzde mobil uygulamaların yaygınlaşması ve buna bağlı olarak veri trafiğinin artmasıyla birlikte, bu verileri hızlı ve yüksek kaliteli bir şekilde sunmak kablosuz hücresel ağlar için giderek zorlaşmaya başlamıştır. Bu talebi karşılamak için, çoklu-erişimli uç hesaplama (MEC) olarak adlandırılan, bulut sunucuların bilgi işlem yeteneklerini uç noktalarda da sağlayarak mesafeye dayalı yüksek gecikmeyi ve veri trafiğini azaltma imkanı sunan bir model oluşturulmuştur. Bu modelin önemli bir uygulaması da, verilerin hızlı dağıtım amacıyla uç noktalar olarak tanımlanan baz istasyonlarının önbelleklerinde depolanması işlemidir. Bu noktada, artan veri trafiğinin özelikle sosyal medya uygulama kullanıcıları tarafından yaratıldığının görülmesi, bu uygulamaların performansını iyileştirmek için baz istasyonlarında etkili depolama tekniklerinin araştırılmasına yol açmıştır. Bu tezde, mobil kullanıcılar arasındaki sosyal etkileşimin ağdaki veri dağıtım modelleri üzerinde güçlü bir etkiye sahip olduğu gerçeğinden hareketle, sosyal gruplara bağlı bir uç önbellekleme sistem modeli önerilmektedir. Kullanıcıların ilgi alanlarına göre manuel olarak gruplandırıldığı literatürdeki diğer çalışmalardan farklı olarak bu çalışma, sosyal grupların kullanıcılar tarafından oluşturulduğu ve üyelerin aynı ilgi alanlarını paylaşmak zorunda olmadığı bir sisteme dayanmaktadır. Bu yöntemle sosyal grup üyelerinin konumları ve bu üyelerin ilgili uygulamaları kullanma istekliliği göz önünde bulundurularak, verileri etkili bir şekilde uçlarda depolayıp, kullanıcılar tarafından deneyimlenen kalitenin arttırılması ve mesafeye dayalı yüksek gecikmenin azaltılması hedeflenmektedir. Önerdiğimiz yöntemlerin performansları kapsamlı deneylerlerle değerlendirildiğinde, yöntemlerimizin gecikmeyi ve ağ yükünü önemli ölçüde azalttığı görülmektedir.

Anahtar sözcükler: 5G ağları, çoklu-erişimli uç hesaplama.

Acknowledgement

First of all, I give my deep appreciation to my advisor Prof. Dr. İbrahim Körpeoğlu. I am very grateful for his continuous support and patience. It wouldn't be possible for me to conduct this study without his constant guidance and encouragement.

I would like to thank the members of the committee, Prof. Dr. Özgür Ulusoy and Prof. Dr. Ertan Onur, for sparing the time to evaluate this work and their valuable feedback. I also thank Vodafone and Information and Communication Technologies Authority of Turkey for supporting this research within the framework of the 5G and Beyond Joint Graduate Support Programme.

I would like to thank to Osman Emre Deniz who provided a great help for this work. Without his contribution, this work would not have been completed.

I also would like to thank Ozancan Doğan. I am grateful for his friendship, support and guidance since high school.

I also thank Canberk Duman for always supporting me and always being by my side. Without him, keeping up with all the struggles would be a lot harder. I am also very grateful for his invaluable feedbacks.

Lastly, I owe the most sincere thanks to my parents Hayrullah and Nur, and my sister Alkım. Their unconditional and endless love and support is always my motivation to go forward. To them I dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Developments in communication technologies have led to the explosive growth of mobile connectivity over the past years. Currently more than 60% of the world's total population use the Internet while Cisco estimates that this ratio will increase to 70% by 2023 and the number of mobile connections will grow to 13.1 billion [1, 2]. With the increasing popularity of social media and media sharing services such as Facebook, Instagram, YouTube, etc., the largest consumer of global Internet usage become social media users. Studies show that users are spending more than one-third of their online time on social media platforms with a daily average of 2.5 hours [3]. Similarly, being one of the most popular social media platforms, YouTube has over 5 billion videos being watched by 30 million visitors with an average of 875,000 new users each day in 2019 [4].

This massive interconnection of devices and people creates huge growth in mobile data traffic, degrading the quality of service (QoS) perceived by the users [5, 6, 7]. The International Data Corporation suggests that, collective sum of the world's data traffic will grow from 33ZB to a 175ZB by 2025, for a compounded annual growth rate of 61 percent [8]. Correspondingly, mobile network operators are struggling to cope with the significant rise in the data consumption. According to their statistics, average monthly media consumption on mobile devices exceeds 4GB in the US, with video an ever increasing percentage of that [9]. Such an

increase in mobile media demand causes 33% of users to face poor streaming quality and creates a big concern on data collection and storage [10].

To overcome these challenges, European Telecommunications Standards Institute (ETSI) introduced Multi-access Edge Computing (MEC) at the edges of the wireless networks to reduce delay, data and tasks exchange between edge devices and remote data centers [11]. As serving content from distant servers causes long delays and decreases quality of experience (QoE), MEC brings communication, computation, caching, and control closer to mobile users by deploying servers at the edge of a network, i.e., on base stations (BSs) of a wireless network [12, 13]. Considering the fact that the majority of data traffic consumption is originated from the requests of different users for the same popular contents, caching can be also beneficial for MSNs by reducing access latency. However, compared with centralized cloud computing, MEC servers have limited resources for handling data at the edges [14]. Therefore, when MEC is used in a network for caching, using edge server storage efficiently and effectively becomes an important problem.

There are several studies about how MEC should be deployed and used in cellular networks [6, 15, 16, 17, 18]. A range of concerns such as delay, energy efficiency, traffic load, or joint optimization of them can be addressed by appropriately choosing at which servers or data centers to place which piece of data, given a group of servers or data centers that reside at different locations [15, 16, 17, 18]. The main idea is to place popular content into edge servers so that most of the requests can be served from local caches of base stations, instead of centralized servers, which are reached via bandwidth-limited backhaul links [19].

However, there are some critical challenges for placing the data of MSNs over clouds and servers. In social networks, social connectivity and interaction among connected users have a huge influence over data delivery patterns. Thus, the knowledge of social network features can be leveraged to use cache resources efficiently and to provide better quality of experience to users in accessing contents. In this respect, since the users of MSNs are interconnected, placements of their data should be interdependent [20, 21, 22, 23]. While choosing the best location

for a user's data, considering the information of that user alone will not be sufficient. We must also consider other users who will access it. Each user is not independent and therefore cannot be treated separately when it comes to data placement. This placement strategy is unlike the conventional Internet services where users may not need to be jointly considered.

Prior socially-aware edge caching strategies for MSNs mainly focus on dividing users into different communities based on their mutual interests and intimacy. These studies also assume that users in the same community tend to share data among each other due to their large interest similarity [24, 25, 26]. Nonetheless, in real-life, all users that are part of the same social media group may not necessarily have a common interest in all subjects. Depending on the content, their frequency of interaction and communication can vary. In particular, social media group members' amount of interaction to a content may not necessarily be the same, which leads to varying levels of user engagement among the members of a social media group for the same content. Unlike other studies where users are manually grouped according to their interests, our approach is based on user-defined social groups where users in the same group are neither obligated to share the same interest nor be attentive to the shared content. Consequently, users' social relationships, their different interests, and locations are jointly considered in our work.

In this thesis, we propose a socially-aware edge caching system model for MSN applications considering social groups of users. Our system caches files into base stations considering the locations of social group members and the willingness of them in accessing the files. The goal is to reduce latency and network load. We propose a socially-aware edge caching algorithm where a batch of files are jointly considered and placed in the base stations in a region. We also propose a dynamic and incremental algorithm where files are placed one by one as they arrive. The algorithm uses a replacement based recursive approach to perform better placement. We conducted extensive simulation experiments to evaluate the performance of our proposed methods and identify the cases where they are especially useful. Our results show that our methods are reducing latency and network load significantly.

The remainder of this thesis is organized as follows. In Chapter 2, we present the related work. In Chapter 3, we formally define the problem as a mathematical program, and present some preliminaries. Additionally, we present our proposed algorithms and their bound analysis in detail. In Chapter 5, we report and discuss the results of our extensive performance experiments. Finally, in Chapter 6, we give our conclusions.

Chapter 2

Related Work

During the last decade many studies have been carried out to reduce traffic load on mobile networks and increase QoE for users considering the large data consumption of social networks with respect to the overall traffic. [27] is one of the first studies on potential performance benefits of edge caches. The objective of the study was to determine how much of the access traffic is served by edge caches as opposed to the centralized data center. After tracing the traffic distribution of Facebook, they revealed that edge caches served more than 89% of requests for the most popular images. Similarly, [28] surveyed variety of resource allocation methods in fog radio access networks. A number of valuable findings emerged from their work including the fact that caching popular social media at the edges of the network provides high spectral efficiency while maintaining low latency.

[29, 30] worked on caching potentials of YouTube platform. These studies compared serving videos from the centralized and edge data centers and reveal that about 40% of requests can be delivered from a cache. Similarly, [31] utilized a caching framework to effectively distribute social media videos and showed that high levels of requests are served by local edge servers (58.7%) instead of centralized servers.

Considering the studies [27, 28, 29, 30, 31], it can be seen that edge caches

can significantly decrease the amount of download traffic and achieve good hit rates. In this respect, to reduce network load further and improve the hit ratio of content sharing, effective management of cache resources in MSN applications is an important research problem.

Leveraging mobility patterns of users in the network, [32] proposed a cooperative caching mechanism for MSNs. To increase content access rate, this mechanism placed content only in centralized users who can be easily accessed. However, this approach increases the caching load of users in these areas and degrades the network performance significantly. Similar to [32], [33] proposed a proactive edge caching model where each user can either access the content from BSs or receive it from the other users via data sharing. As such, content sharing mechanism relies on the social tie strength between the users. However, as the mechanism in [33] transmitted the content to a set of users with large numbers of social relationships and entrust them to share it with others, it may result in an inaccurate estimation of a users' willingness to receive the content, leading to a failure to fully utilize the limited cache resources. Besides, considering the effect of mobility on data sharing, their mechanism adaptively adjusts the optimal set of users to deliver the content, which brings extra computational load to the network. These approaches are significantly different from ours in that they are not considering virtual communities. It is essential to analyze mobile user behaviors for data sharing from social perspectives to improve success ratio.

Considering the fact that users with the same or similar interests will frequently form a social group, some studies provide social group aware caching mechanisms for MSNs. Based on the close geographical relationship among users, [34] divided users into different groups where users within the transmission range of each other can exchange messages. In this mechanism, messages have certain priorities and they are transferred according to their priority order. However, as users from different groups can face unstable connections, low priority messages are not guaranteed to be delivered with this mechanism which affects the success ratio of content sharing. Besides, users geographically far away from each other are not able to share contents, which eventually decreases the network performance as the same content will be cached in several different places. Taking social factors

into account, [35] analyzed how to cache a part of a video on edge nodes for efficiently serving multiple social groups in a mobile social network. Because the cache capacity is limited, the representative mobile users from each social group compete for the caching space provided by the cache node. The group members, which have similar hobbies or interest in the same content, are assumed to access the same video resulting in an inaccurate estimation of a user's social behavior. Unfortunately, these studies fail to fully utilize the limited cache capacities as they neglect the willingness of members in accessing the content.

[36] designed a user cooperative caching mechanism considering various factors that may affect the content sharing behaviors of users in MSNs, including their attributes, community, mobility, and social connections. Their work resembles to ours in consideration of willingness of users in the same community for sharing and receiving content. However, their content caching algorithm only considers similar interests groups, neglecting social groups with heterogeneous interests. Necessary conditions to create a social group are not only shared interests, but also values, representations, and social bonds. Thus, members of social groups do not necessarily have similar interests.

In order to increase the hit probability and mitigate network congestion in a caching system, one must jointly look at heterogeneity of interests in a social group and willingness of group members in accessing content. Thus, different from aforementioned studies, we design a socially aware caching mechanism considering various factors that may affect the content sharing behaviors of users in MSNs including varying interests within groups, social attributes, and storage and transmission capacities of edge servers.

Chapter 3

System Model and Problem Formulation

This thesis proposes a comprehensive socially-aware edge caching framework to optimize intra-network traffic and QoE in cloud-based MSNs, while ensuring storage and transmission capacities of edge servers.

Without loss of generality, we first define the properties of a wireless edge network environment for which our solution can be applied. Then, in Section 3.2 and 3.3, we provide the details of our LP solution.

3.1 System Model

Assuming that servers and MSN users are all geographically distributed, we are concentrating on caching in one edge network that will be covering a region of interest, such as a city or town, and the whole mobile cellular network in a country is a collection of such edge networks. Each user belongs to one or more social groups, generates files, and wants to share them with his or her social groups. We expect that members of these social groups would be interested in accessing

these files, and therefore, files need to be cached considering the locations of the members.

An edge network consists of a multi-hop edge-core network and a set of base stations (BSs). The edge-core network, which we will just call core network in the rest of the thesis, connects the base stations together. We assume there is a set of N base stations, denoted with $\mathcal{N} = \{1, 2, ..., N\}$, connected with the core network, where $n \in \mathcal{N}$ represents the n-th base station. We also assume that there is a set of U user elements (UEs), denoted with $\mathcal{U} = \{1, 2, ..., U\}$, randomly located in the area covered by these base stations. The coverage of two base stations may overlap. Each base station is connected to the core network and the core network is connected to the rest of the mobile cellular network, which includes centralized data centers. We assume that base stations can store and serve content, i.e., act as cache nodes.

A user is connected to only one base station at a time. A base station can only serve the users who are connected to it. Let $C = [c_{un}]$, $u \in \mathcal{U}$, $n \in \mathcal{N}$, be a binary matrix which denotes whether user u is connected to base station n or not.

There is a set of F files, denoted with $\mathcal{F} = \{1, 2, ..., F\}$, generated by users and that can be cached. Each file $f \in \mathcal{F}$ can be cached in one base station at most. The size of a file f ($f \in \mathcal{F}$) is denoted by s_f.

An example system is provided in Figure 3.1.

We assume that each user belongs to at least one social group and groups can have multiple common members. Let $\mathcal{G} = \{1, 2, ..., G\}$ be the set of all social groups. The binary matrix $Sg = [sg_{ug}]$, $u \in \mathcal{U}$, $g \in \mathcal{G}$ denotes whether user u belongs to social group g. This information can be provided to the edge caching system by the social media application service provider. We assume that the relation between the files, their creators, and their corresponding social groups is also provided to the edge caching system by the service provider. The information that a file f being created by a user u and wanted to be sent to a social group g is known and denoted by $[b_{fug}]$. A binary-valued 3D matrix $B = [b_{fug}]$, where

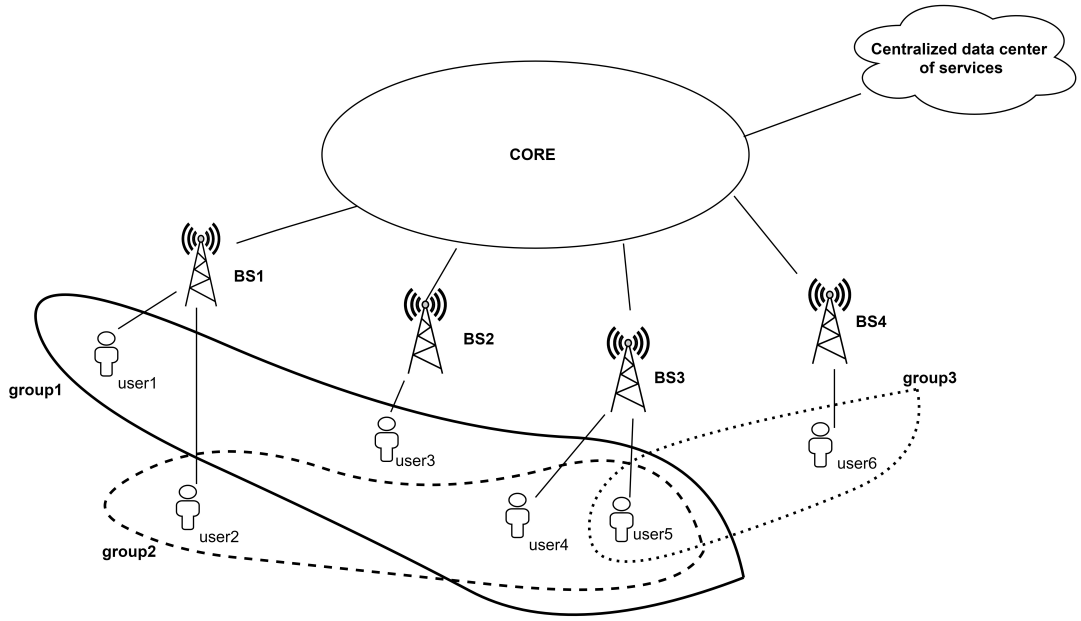


Figure 3.1: Example system.

$f \in \mathcal{F}$, $u \in \mathcal{U}$, and $g \in \mathcal{G}$, gives this information for all files, users, and groups. Our goal is to decide which base station n should cache the file f so that for the participants of the selected group g, both latency and network load are minimized.

The frequency of the MSN application usage of each user is also important. If the frequency of application usage of a user is high, then our algorithm will prefer to place the content in a base station close to that user considering the high access rate of the user for the content. Thus, the frequency of application usage of users is taken into account while placing content on the base stations. Let a_u be the frequency of application usage of user u and let $A = [a_u]$ $u \in \mathcal{U}$, be a vector that represents the frequency of application usage for all users.

Let $D = [d_{nm}]$, $n, m \in \mathcal{N}$, be a symmetric matrix denoting the hop-distance between base stations n and m. Given two base stations n and m, the shortest hop distance between them is calculated and assigned to the corresponding element of the D matrix. The diagonal elements of the D matrix are always 0.

When a file is not cached, it is stored in the centralized server of the social media application provider, outside of the edge network. Hence, when a user requests a

file that is not in the cache, the file has to be downloaded from the center. This will cause more delays compared to downloading from a caching base station inside the edge network. We will represent this delay again with the number of hops of the download path. Let H be the number of hops between a user and the central server of the MSN application. We assume that $H > D_{max}$, that means $H > d_{nm}$ for all $n, m \in \mathcal{N}$. For simplicity, we assume all users in the edge network will experience the same delay H if the file is downloaded from the center.

We use a boolean decision variable $x_{fn},\ f \in \mathcal{F}$ and $n \in \mathcal{N}$, to present the file caching status as:

$$x_{fn} \begin{cases} 1 & \text{if file } f \text{ is cached in base station } n \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Let $Nc = [nc_n]$, $n \in \mathcal{N}$ denotes the caching capacity of base station n. If a file f is placed in base station n, then nc_n will decrease by the size of that file, namely s_f.

All parameters of our system model are presented in Table 3.1.

Table 3.1: System Parameters.

Symbol	Description
F	Total number of files
U	Total number of users
N	Total number of BSs
G	Total number of groups
s_f	Size of file f
sg_{ug}	1 if user u is a member of social group g, 0 otherwise
b_{fug}	1 if file f is generated by user u and to be sent social group g, 0 otherwise
c_{un}	1 if user u is connected to BS n, 0 otherwise
a_u	Frequency of application usage of user u
d_{nm}	Hop-distance between BS n and BS m (hop-distance per unit traffic load)
nc_n	Storage capacity of BS n
H	Hop-distance between a user and centralized server
x_{fn}	1 if file f is cached in BS n, 0 otherwise

3.2 Minimizing Total Latency

Taking the parameters defined in Section 3.1 into consideration, we denote the latency (L) as follows:

$$L = \begin{cases} b_{fug}\ sg_{kg} a_k c_{km} d_{nm} x_{fn} & \forall f \in \mathcal{F}, g \in \mathcal{G}, n \in \mathcal{N}u, k \in \mathcal{U} \\ H\ b_{fug} sg_{kg} a_k \prod_n (1 - x_{fn}) & \forall f \in \mathcal{F}, (u,k) \in \mathcal{U}, g \in \mathcal{G}, n \in \mathcal{N}u, k \in \mathcal{U} \end{cases} \tag{3.2}$$

The first line of (3.2) corresponds to the case when a file could be cached in one of the base stations and accessed from this cache. More specifically, it gives the latency when a file f generated by user u to be sent to social group g, is cached in BS n and is requested by a user k who is connected to BS m in the same social group g. The terms of this part of the equation are:

- b_{fug} parameter shows that a file f owned by user u is sent to social group g,
- x_{fn} is the caching status of file f in BS n,
- sg_{kg} shows user k in the same social group g, who might request f,
- c_{km} is user k's base station m,
- d_{nm} is the hop-distance between BS m and BS n, and
- a_k is the frequency of application usage of user k.

The second line of (3.2) corresponds to the case when a file f could not be cached in any of the base stations and therefore retrieved from the center. The terms of this part of the equation are:

- H is the hop-distance between a user and the centralized server,

- b_{fug} shows that a file f owned by user u is sent to social group g,
- sg_{kg} shows user k in the same social group g, who might request f,
- a_k is the frequency of application usage of user k, and
- $\prod_n(1 - x_{fn})$ ensures that file f is not cached in any of the BSs and should be downloaded from the centralized server.

Then, total latency (L_{total}) is given by:

$$L_{total} = \sum_f \sum_g \sum_u \Big[\sum_k (b_{fug} sg_{kg} a_k c_{km} d_{nm} x_{fn}) + \sum_k \big(H b_{fug} sg_{kg} a_k \prod_n (1 - x_{fn}) \big) \Big] \tag{3.3}$$

Thus, the social-aware caching optimization problem with latency costs can be defined as:

$$\min \quad L_{total} \tag{3.4}$$

$$\text{subject to} \quad \sum_n x_{fn} \leq 1 \quad \forall f \in \mathcal{F}. \tag{3.5}$$

$$\sum_f x_{fn} s_f \leq n c_n \quad \forall n \in \mathcal{N}. \tag{3.6}$$

$$x_{fn} \quad is \quad binary \quad \forall f \in \mathcal{F}, n \in \mathcal{N} \tag{3.7}$$

The objective function (3.4) represents the total latency experienced by all users. The constraint in (3.5) ensures that content can be stored only in one place. The constraint in (3.6) imposes cache storage capacities.

3.3 Minimizing Total Network Load

Similar to social-aware caching optimization problem with latency costs, assuming that traffic overhead occurred while establishing a link is very small and can be neglected, the social-aware caching optimization problem with load costs can be formulated as follows.

1. The network load incurred while bringing a requested content from base station n to base station m is given by:

$$NL_1 = \sum_f \sum_g \sum_u \sum_k b_{fug} x_{fn} sg_{kg} a_k c_{km} d_{nm} \tag{3.8}$$

2. The network load incurred while bringing the requested content from the centralized server, which occurs when content is not cached in any of base stations, is given by:

$$NL_2 = \sum_f \sum_g \sum_u \sum_k H b_{fug} sg_{kg} a_k \prod_n (1 - x_{fn}) \tag{3.9}$$

In Equation (3.9), H represents the expected load per request when content is retrieved from the center. Like delay, the load is also assumed to be proportional to the number of hops between the content requester and centralized server. Therefore, we use H, i.e., the number of hops between the requester and center, to model the load incurred on the network per request. We assume that $H > d_{nm}$ for all $n, m \in \mathcal{N}$. That means $H > D_{max}$.

Then, total network load can be expressed as follows:

$$
\begin{aligned}
NL_{total} &= NL_1 + NL_2 \\
&= \sum_f \sum_g \sum_u \sum_k \Big(b_{fug} x_{fn} s g_{kg} a_k c_{km} d_{nm} \\
&+ H b_{fug} s g_{kg} a_k \prod_n (1 - x_{fn}) \Big)
\end{aligned}
$$

Thus, we can formulate the social-aware caching optimization problem with load costs as:

$$
\begin{aligned}
\min \quad & NL_{total} && (3.10)\\
\text{subject to} \quad & \sum_n x_{fn} \leq 1 \quad \forall f \in \mathcal{F} && (3.11)\\
& \sum_f x_{fn} s_f \leq n c_n \quad \forall n \in \mathcal{N} && (3.12)\\
& x_{fn} \quad is \quad binary \quad \forall f \in \mathcal{F}, n \in \mathcal{N} && (3.13)
\end{aligned}
$$

Both of the social-aware caching optimization problems are NP-hard which can be shown using a reduction from the minimum vertex cover problem. We next present the proof for the NP-hardness of our problems.

Theorem 1. *The* SOCIAL-AWARE CACHING OPTIMIZATION PROBLEM *is NP-hard.*

Proof. The proof uses a reduction from the minimum vertex cover problem. Given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a vertex cover is a subset $S \subseteq V$ in which each edge in G has at least one endpoint in S. The minimum vertex cover problem is to find the minimum size of the vertex cover in a graph [37].

We give a reduction from the minimum vertex cover problem as follows. Without the loss of generality, suppose that for each content $f \in F$, we have a weighted

undirected graph $G = (V, E, W)$, where V is the set formed by the elements of recipient users $A \subseteq U$ and their affiliated BSs $B \subseteq N$, and E is the set of the links among $v_i, v_j \in V$ with a weight of $w_{e_{v_i,v_j}}$. To be more clear, $(A, B) \subseteq V$, $E \rightarrow\{\mathrm{e}_{v_i,v_j} | ((v_i, v_j) \in B^2$ and $i \neq j)$ or $(v_i \in A$ and $v_j \in B)\}$ and $W \rightarrow\{\mathrm{w}_{e_{v_i,v_j}} | d_{v_i,v_j}$ if $(v_i, v_j) \in B^2$ or a_{v_i} if $v_i \in A$ and $v_j \in B\}$.

The social-aware caching optimization problems tries to determine the optimal caching strategy with a minimal cost. That is, for each f, it determines a vertex set $S \subset V$ where $S = B$ and calculate the total cost for each element of S using edge weights. Then, it processes the obtained costs for all files concurrently to be able to detect feasible and optimal caching for the whole system with the minimum total cost. When we consider these steps, it is not hard to see that calculating total cost for each element of S takes linear time in the size of the graph, i.e. $\mathrm{O}(|E|)$ time. Similarly, processing the obtained costs for all files takes $\mathrm{O}(|V| + |E|)$ time. We also observe that subset S is indeed the minimum vertex cover of G. Thus, since the minimum vertex cover problem is NP-hard [38], the social-aware caching optimization problems are also NP-hard.

□

3.4 Summary

In this chapter, we have presented a linear programming solution for our social-aware caching optimization problem.

Assuming that servers and MSN users are all geographically distributed, we are concentrating on an edge network system consisting of a multi-hop edge-core network and a set of base stations. In that system, each user belongs to one or more social groups, generates files, and wants to share them with his or her social groups. By using the available storage resources of BSs, we want to decrease both storage and load burdens on centralized data centers. To do that, we formulate two caching optimization problems, one optimizing latency and the other

optimizing network load.

We also show that both of these social-aware caching optimization problems are NP-hard. We prove this by using a reduction from the minimum vertex cover problem.

Chapter 4

Proposed Algorithms

Finding an optimal solution using the linear programs (3.4) and (3.10) is very difficult and time-consuming for large parameter values. Therefore, we propose a heuristic algorithm to solve the social-aware caching optimization problem. Since a set of files are jointly considered, we call this algorithm as Batch Cache Placement Algorithm (BCPA).

We also propose a dynamic and incremental heuristic algorithm, where incoming files are treated on an individual basis and placed one by one as they arrive. While placing a file into the cache system, we may remove an existing file in a candidate base station to make space, if this would decrease total latency. We use a similar strategy to place the removed file. We call this recursive algorithm as Dynamic Cache Placement Algorithm (DCPA).

In Section 4.1, we provide some preliminaries for our algorithms. In Section 4.2 and 4.3, we present our heuristic algorithms, BCPA and DCPA, and their bound analysis. In the subsequent section, Section 4.4, we provide fairness analysis of the algorithms.

4.1 Preliminaries

As will be shown later, a new matrix $W = [w_{gn}]$, $g \in \mathcal{G}$ and $n \in \mathcal{N}$, is introduced for the suggested algorithms. For a file f that can be cached in a base station, the matrix W denotes total latency experienced by the members of each social group. Specifically, w_{gn} shows the total propagation delay when the content is being served from a base station n to social group g. The necessary steps to construct the W matrix and a numerical example is provided below.

Assume that $U = 6$, $N = 4$, $G = 3$, and A, S_g, C, and D matrices are as follows:

$$A = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} \quad Sg = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 2 & 3 & 4 \\ 2 & 0 & 1 & 2 \\ 3 & 2 & 0 & 2 \\ 4 & 3 & 2 & 0 \end{bmatrix}$$

1. Firstly, we perform an element-wise multiplication of A and Sg matrices and construct a weighted $Sa_{U \times G}$ matrix representing user-social group associations as well as their application usages. The elements of the resulting matrix are nonzero if user u belongs to social group g. Each nonzero $sa_{u,g}$ element denotes the frequency of application usage of user u. An example for this step is provided below.

$$A \odot Sg = Sa$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} \odot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 3 & 0 & 0 \\ 4 & 4 & 0 \\ 5 & 5 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

2. Then, we multiply the transpose of Sa matrix obtained above with C matrix. The result is $Sc_{G \times N}$ matrix which is a weighted association of social groups and base stations showing the total application usages of group members who are connected to base station n. An example for this step is provided below.

$$Sa^T C = Sc$$

$$\begin{bmatrix} 1 & 0 & 3 & 4 & 5 & 0 \\ 0 & 2 & 0 & 4 & 5 & 0 \\ 0 & 0 & 0 & 0 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 9 & 0 \\ 2 & 0 & 9 & 0 \\ 0 & 0 & 5 & 6 \end{bmatrix}$$

3. Lastly, we multiply Sc and D matrices and construct $W_{G \times N}$ matrix. Each element in this matrix shows total delay experienced by the members of social group g when the content is cached in base station n. For multiple social groups receiving the same content, respective columns of W matrix will be summed. An example for this step is provided below.

$$ScD = W$$

$$\begin{bmatrix} 1 & 3 & 9 & 0 \\ 2 & 0 & 9 & 0 \\ 0 & 0 & 5 & 6 \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 21 & 10 & 5 & 18 \\ 18 & 11 & 4 & 15 \\ 28 & 17 & 6 & 5 \end{bmatrix}$$

It is important to understand the above steps before proceeding further. Given the receiving social groups, we search w_{gn} values and select an available base station with the smallest possible latency. For example, in the above example, caching in BS3 for social group1, BS3 for social group2 and BS4 for social group3 will give the smallest latency. Then, total latency L_{total} will be the sum of these selected w_{gn} values and delay for files that are stored in the centralized servers. During this process, if enough storage is not available on the BS giving the lowest latency, a new BS with the second lowest latency will be selected.

As pointed out before, we assume that common members of multiple social groups will request the same file multiple times since our model is not relying on mobile devices to cache the accessed files. For that case, we sum respective columns of W and count the latency experienced by common members multiple times to find L_{total}. For example, in the above example, if only social group1 and group2 will receive the content, caching in BS1, BS2, BS3 and BS4 have a total latency of 39, 21, 9 and 33, respectively.

Time complexity of the above steps is $\mathcal{O}(UN)$, if $GN \leq U$, and $\mathcal{O}(GN^2)$, otherwise.

4.2 Batch Cache Placement Algorithm

To achieve a low-complexity solution, we propose an iterative heuristic algorithm called Batch Cache Placement Algorithm (BCPA).

Let $m \leq F$ be the batch size, i.e., the number of files in each set. The objective of BCPA is to cache files in batches. To do that, we group consecutive files

together into batches of size m. Then, we sort files in each batches into priority order by their number of recipients. This is because, we aim to place them by their potential number of accesses, i.e, to be able to cache files with the highest requests first. Compared with an arrival order based caching, this method utilizes the limited storage capacity of BSs and reduces total latency. Specifically, sorting can be done by using one of two criteria: i) total number of recipient groups, ii) total number of recipient users. The performance comparison of these two criteria is given in Section 5.2.

The algorithm follows an iterative placement strategy. In the Algorithm 1, shown below, files are divided into batches and sorted by their number of recipients. Then for each file in a batch, Algorithm 2 selects an available base station that gives the smallest latency using the W matrix. That is, for each file, Algorithm 2 finds and returns an available base station with the lowest latency. To do that, it first sorts the rows of W matrix in ascending order based on access latency and returns the sorted elements as W^* matrix. Then, starting from the first element of W^* matrix, it checks the availability of base stations and returns the first available base station which gives also the lowest latency. When there is not enough space in the system to cache the given file, it will be stored in the centralized server of the social media application provider.

After returning from Algorithm 2, Algorithm 1 calculates the total latency by summing up the obtained latency and previous latency values and decreases the storage capacity of the respective BS.

The above steps are repeated for each batch. Thus, time complexity of BCPA algorithm is $\mathcal{O}(F(\log F + G + U + N \log N))$. The pseudo-code of BCPA is given in algorithm 1 and Algorithm 2.

Note that BCPA algorithm does not need to have all files arrived at the time of placement. It waits until a batch of files to arrive and then starts placement. After a batch is placed, it waits for the next batch to arrive, and so on.

Clearly, BCPA's efficiency in caching content greatly depends on the batch size

Algorithm 1 BCPA $(m, F, A, NC, W, S, H, D)$

Q: Split F into batches of size m sequentially
for each batch $q \in Q$ **do**
  Sort q in descending order with respect to their total number of recipient groups/users
  for each file $f \in q$ **do**
    if all BSs are full in capacity, i.e, f can not be cached **then**
      $L_{total} += a_u H$
    else
      w^*, i = SelectBS(NC, W)
      $nc_i -= s_f$
      $L_{total} += w^*$
    end if
  end for
end for

Algorithm 2 SelectBS (Nc, W): Finds a base station with minimum cost

Ensure: w_i^*,i
$i = 1$
$W_{1XN}^* \longleftarrow$ Sort W matrix
while $i < N$ **do**
  if BS i has enough capacity for caching **then**
    Return total cost and index of BS i by w_i^*,i

  end if
  i++
end while

and the arrival order of files. That is, whether the popular contents are arrived earlier or not is an important issue for the performance of BCPA. To overcome these drawbacks of BCPA, we propose another algorithm called Dynamic Cache Placement Algorithm where a replacement based recursive approach is used.

4.3 Dynamic Cache Placement Algorithm

We also propose Dynamic Cache Placement Algorithm (DCPA) where incoming files are treated on an individual basis and cached as soon as they arrive. With this

algorithm, we provide a replacement based recursive solution, where a previously placed file may be replaced if that would result in lower latency. The pseudocode of DCPA is given in Algorithm 3 and Algorithm 4.

In Algorithm 3, we build an $W^*_{1\times N}$ matrix. This matrix is obtained from W matrix and denotes the sorted total latency of the file for each BS. This step is necessary for simplification, and obtained matrix will be used by the replacement algorithm. After sorting W^* in ascending latency order, we provide it to Algorithm 4.

In Algorithm 4, we recursively search for optimal base station (i.e., the base station that has minimum latency in W^*). If the best possible base station for file f is not available, existing files in that base station will be investigated. If any of these previously placed files f_p has a lower number of recipient social groups than file f, we replace these two files. That is, we may remove an existing file f_p in a candidate base station to make space for file f, if this would decrease total latency. Then, similarly, a best possible base station for f_p is searched using algorithm 3. Here, we ignore the delay and network load incurred due to replacement.

The time complexity of the DCPA algorithm is $\mathcal{O}(N \log N + FN)$.

Algorithm 3 DCPA (f, W, $L_{total}, A, H, N, NC, S$)

Ensure: L_{total}
1: Build $W^*_{1\times N}$ matrix from number of recipients
2: $W^*_{1XN}, I_{1xN} \longleftarrow$ Sorted W^* matrix and BS index array
3: PlacementRec(f, I, W, W^*, i=1, $L_{total}, A, H, N, NC, S$)

Compared to BCPA method, DCPA approach imposes additional computational burden on the server, especially for the large number of BSs and files. One approach to deal with this burden is to use coded caching where several requests can be satisfied with a single transmission [39, 40]. This results in lower computational load at the expense of higher rates over the shared link.

Algorithm 4 PlacementRec($f, I, W, W^*i, L_{total}, A, H, NC, N, S$): Recursive Algorithm to find best possible BS

Ensure: L_{total}
If all BSs are checked:
if i==N **then**
  File can not be cached and find total latency by $L_{total} += a_u H$
else
  if $nc_i \geq s_f$ **then**
    $nc_i = nc_i - s_f$
    $L_{total} += w_i^*$
  else
    Find $minFile$
    if $minFile! = f$ **then**
      Remove $minFile$ from BS i and place f:
      $nc_i = nc_i$ -s_f+$s_{minFile}$
      $L_{total} += w_i^*$
      Search another BS for $minFile$:
      DCPA($minFile$, W, $L_{total}, A, H, N, NC, S$)
    else
      Search another BS for f
      PlacementRec(f, $I, W^*, i+1, L_{total}, A, H, NC, N, S$)
    end if
  end if
end if

4.4 Fairness Analysis of Algorithms

Besides optimizing the latency, traffic load and capacity usge of the network, achieving fairness is another important issue which should be considered in caching systems [41]. The basic idea is to design a system where users in the same network have the same chance to access contents they are interested in. Thus, a key question is how to fairly allocate the cache storage resource among different requests.

While there exists different criteria on the fair allocation, in this thesis, we refer to fairness as equal share of the resource being allocated to social groups in our caching strategy [42]. We investigate our proposed algorithms on request-based fairness, associating how many requests of each social group can be served from

caches of BSs. That is, if a social group has much better access performance than others, it will violate the fairness of the system. To measure the fairness among social groups, we use Gini index method [43, 44]:

$$I_{gini} = \frac{1}{2n^2 x^*} \sum_i \sum_j |x_i - x_j| \tag{4.1}$$

where n is the total number of social groups, x_i is the caching ratio of the requests of social group i, and x^* is the average performance of all groups. The index is in the range from 0 to 1, where smaller values imply a more fair system.

Figure 4.1 shows the fairness metrics for BCPA and DCPA. As can be seen from the figure, BCPA performs more fair than DCPA. This is because, in DCPA method, only the contents with the highest requests are cached which leads to users requesting less popular contents have poor accessibility performance than others. In this way, it is hard to ensure the fairness among social groups. However, BCPA's fairness performance is well maintained due to its poor performance in caching. Since BCPA method may not necessarily cache the most popular contents, the fairness among social groups is better preserved.

4.5 Summary

In this chapter, we have presented two heuristic algorithms for caching, since finding an optimal solution to our linear programming formulation is very difficult and time-consuming for large parameter values.

The first algorithm is called Batch Cache Placement Algorithm (BCPA) where a set of files are jointly considered. In this method, consecutive files are grouped together into batches of m. Due to this design, BCPA's efficiency in caching content greatly depends on the batch size and the arrival order of files.

We also proposed a dynamic and incremental heuristic algorithm, where incoming

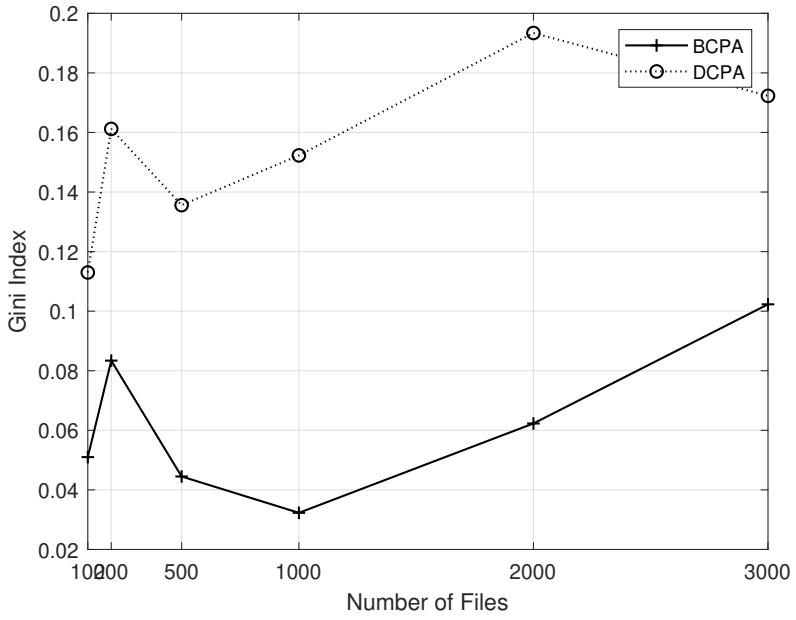


Figure 4.1: Gini index versus number of files.

files are placed one by one as they arrive. In this method, while placing a file into the cache system, we may remove an existing file in a candidate base station to make space, if this would decrease total latency. We use a similar strategy to place the removed file. We call this recursive algorithm as Dynamic Cache Placement Algorithm (DCPA). However, this algorithm imposes additional computational burden on the servers, especially for large number of BSs and files.

In this chapter, we also investigate request-based fairness of the algorithms using Gini index method. We conclude that BCPA performs more fair than DCPA.

Chapter 5

Experimental Results and Evaluation

In this chapter, we present the results of the simulation experiments that we have conducted to evaluate our methods. We characterize the performance improvements offered by our methods over random placement, where files are placed without considering the social groups of users. We use FICO Xpress 8.8 64 bit and MATLAB R2019a 64 bit for the implementations of our linear programming (LP) solution and heuristic algorithms, respectively.

In Section 5.1, we introduce our simulation setup. In Section 5.2, we present our results and compare our methods with each other and with random placement.

5.1 Simulation Settings

We evaluate the performance of our methods under different system model parameters, including the total number of files, total number of base stations, total number of users, total number of social groups, and the total capacity of caches.

To increase the reliability of our simulation results, we repeat each simulation experiment 20 times and report the average. The simulation parameters are listed in Table 5.1.

Table 5.1: Parameter values used in numerical results.

Parameter	Value	
	Simple System	Large System
F	60	1000
U	10	300
N	3	50
G	3	60
H	8 hops	20 hops
m	5 files	250 files
d_{nm}	4 hops	10 hops
s_f	100 MB	
nc_n	1.5 GB	
sg_{ug}	randomly generated	
b_{fug}	randomly generated	
c_{un}	randomly generated	
a_u	randomly generated btw 1-10	
d_{nm}	randomly generated btw 1-10	

Frequency of MSN application usage for some users can be higher than the others. To simulate this fact, we analyze the usage statistics of WhatsApp, one of the most popular mobile social network applications. 54% of WhatsApp users tend to use the application regularly, whereas 38% and 8% of them use it occasionally and rarely. Thus, we right-skewed the values of the A matrix, representing the frequency of usage for different users, in our simulations [45]. Also, the number of users in each social group and the number of files generated by each user are not deterministic; they are randomly generated.

5.2 Simulation Results

In this section we present the results and evaluate our approach. First, we compare the performance of LP formulation, BCPA algorithm, and random placement. For that, we use a simple system with small parameter values. This is because LP formulation takes too much time for large parameter values. Then, we analyze a more complex system with larger parameter values for BCPA, DCPA, and random placement algorithms.

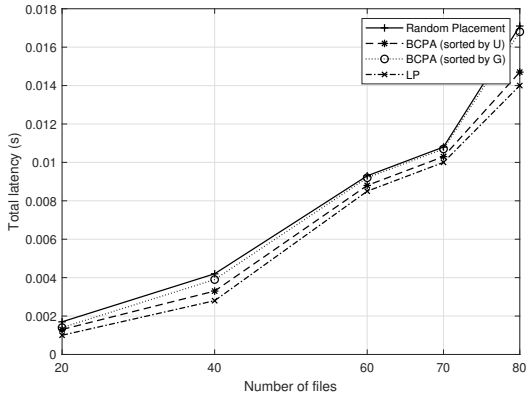
5.2.1 Performance Comparison in a Simple System

The Figure 5.1 shows the delay performance of LP formulation, two versions of our BCPA algorithm (each using one of the two sorting criteria), and random placement, with various values of F, U, G, N. As can be seen from the sub-figures, LP formulation provides the lowest total latency and random the highest. Both versions of the BCPA method provide better network latency than random placement, while approaching to LP formulation in some cases. This is an expected result since LP method makes placement decisions by considering all content together whereas BCPA considers them in separate groups.

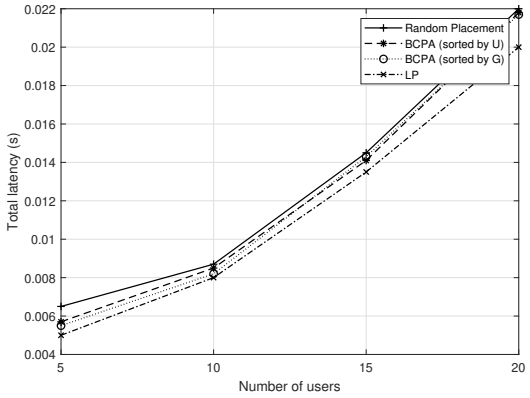
While sharp differences between two methods of BCPA in Figure 5.1a and Figure 5.1c indicate that sorting files by total number of recipient groups is better when placing the content into caches, differences in Figure 5.1d and Figure 5.1b indicate the opposite. Thus, we can not immediately declare a suitable sorting criteria for BCPA from these experiments.

5.2.2 Impact of Batch Size

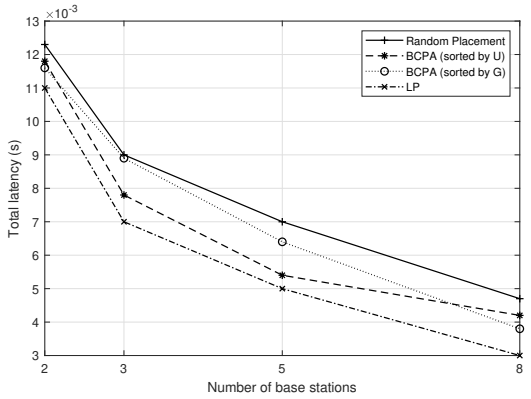
Since BCPA processes a batch of files at a time, we believe that batch size, i.e., the number of files that will be passed through to the network at one time, is an important parameter for delay performance of BCPA method.



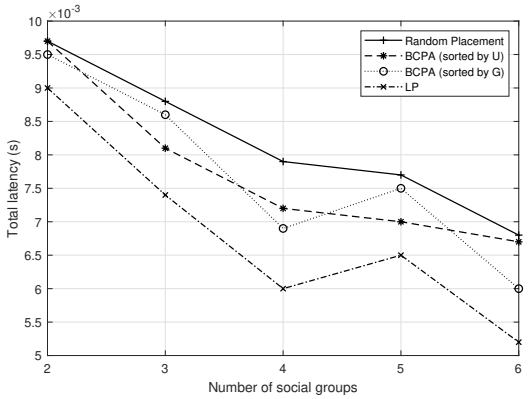
(a) Total latency versus number of files.



(b) Total latency versus number of users.



(c) Total latency versus number of base stations.



(d) Total latency versus number of social groups.

Figure 5.1: Performance comparison of the solutions in a simple system.

Figure 5.2 shows the results of this experiment conducted with file size of 1000 and different batch size values. As can be seen, larger batch size yields lower delay. This is because when the batch size increases, the algorithm is able to see and compare more files before deciding what to cache and therefore ensures more popular items are cached before the storage capacity is filled.

We can also observe that sorting files by their total number of recipient users consistently outperform sorting them by number of recipient social groups. But interestingly, total latency gap between two sorting methods is the largest for batch size of 500. The sharp increase in the gap for large batch sizes indicates that sorting files by groups performs unsatisfactorily. Such high delay performance

difference between two methods is not acceptable for edge caching scheme to be usable in practice. Therefore, we conclude that sorting files by their total number of recipient groups does not perform well and therefore we exclude that method from the rest of our experiments.

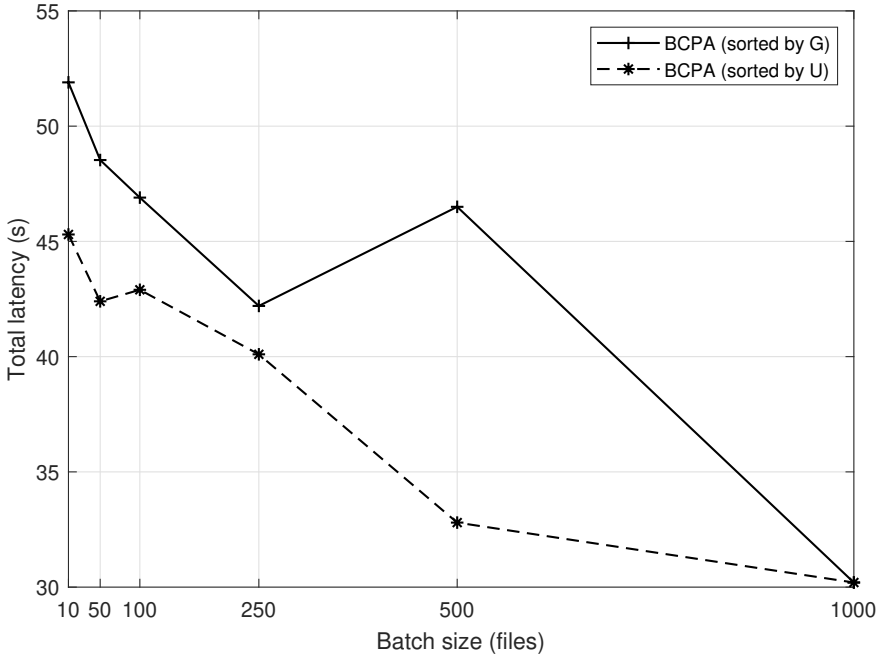


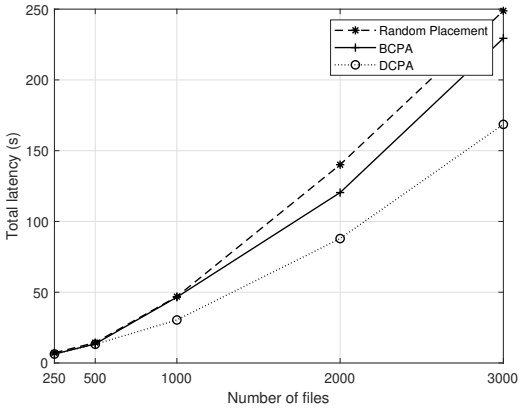
Figure 5.2: Total latency versus batch sizes (m).

5.2.3 Performance Comparison in a Large System

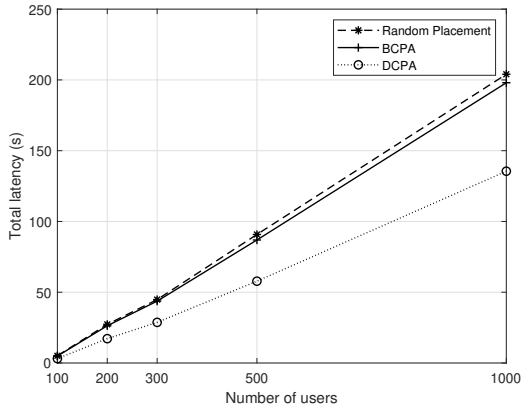
Figure 5.3 shows the behavior of our methods when large parameter values are used. The benefits of DCPA method can be seen clearly from the figure. For all sub-figures, the sharp difference in total latency with DCPA is observed. This is an expected result since towards an optimal solution, DCPA may replace the existing files by repeatedly checking edge caches to see if the total latency can be reduced further by replacing the existing content. From this we can conclude that DCPA method improves the system's delay performance significantly. However, it may incure more overhead on servers.

While discussing the BCPA method, we have mentioned that performance of BCPA greatly depends on the batch size used. As can be seen from the figure,

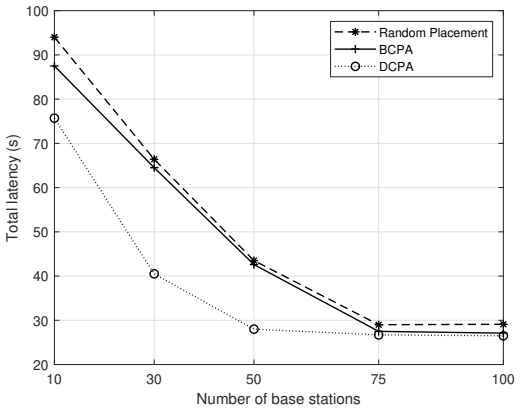
BCPA performs close to random placement, even for a large batch size like 250. Such a high delay is not desirable. This means BCPA is prone to place unpopular contents in edge caches.



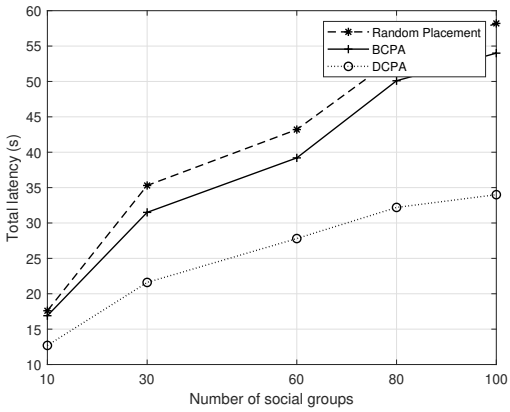
(a) Total latency versus number of files.



(b) Total latency versus number of users.



(c) Total latency versus number of base stations.



(d) Total latency versus number of social group.

Figure 5.3: Performance comparison of the solutions in a large system.

It is important to notice that the graphs in Figure 5.2.3 has a similar pattern as of the Figure 5.2.1. With large file numbers, total latency significantly increases. This is because when the number of files to be cached exceeds total storage capacity of base stations, more files have to be downloaded from the centralized server. The same characteristic applies to Figure 5.3b due to growing demand for files. The total latency, however, is inversely proportional to an increase in

N due to growing total caching capacity of the system. But we notice a different performance trend for parameter G. We can explain this inversely proportional pattern as the more is the number of social groups there is a great chance of having the less number of users per social group, which leads to a decrease in demand for files and decreases the total latency.

But this behavior is not necessarily the case at all times. That is, increased G does not necessarily mean decreased number of users per social group. However Figure 5.3d indicates a direct relationship, where increasing G increases the latency. We can explain this behavior as the affect of number of users on total latency is so dominant that changes in G becomes insignificant. Compared to Figure 5.1d, Figure 5.3d has 3000% more users which may minimize the effect of G, and eventually increases the total latency. Thus, this contrary behaviour of Figure 5.3d may indicate the dominant effect of U parameter.

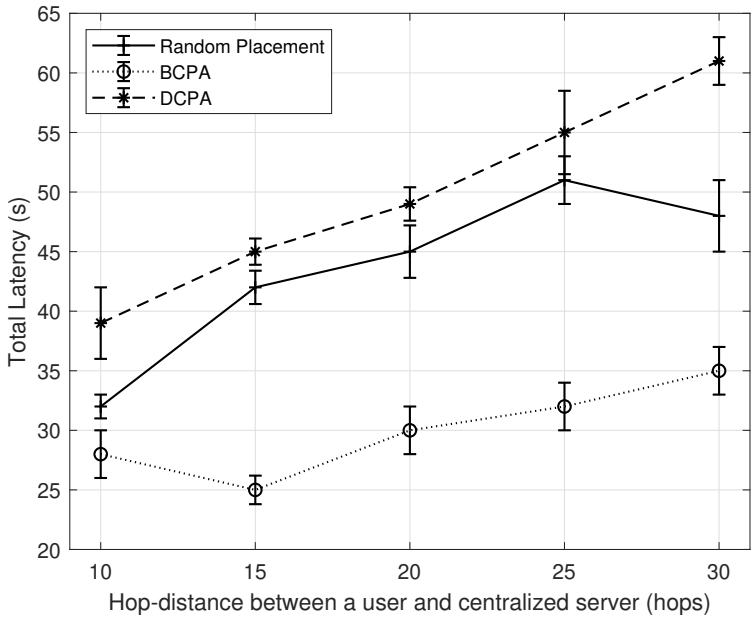


Figure 5.4: Total latency versus hop-distances between users and centralized server (H).

Figure 5.4 evaluates the delay performance of methods with 95% confidence intervals for different hop-distances between users and the centralized server. The values used for H are 10, 15, 20, 25, and 30. For BCPA and random placement

methods, as H increases the total latency increases significantly. With large H values the increase is much sharper. But interestingly, increasing H does not have a significant effect on delay performance of DCPA. Therefore, we can say that performance of DCPA method is near optimal as it stores the least popular content in central servers preventing undesirable increase in total latency with large hop-distances.

5.2.4 Impact of Cache Size

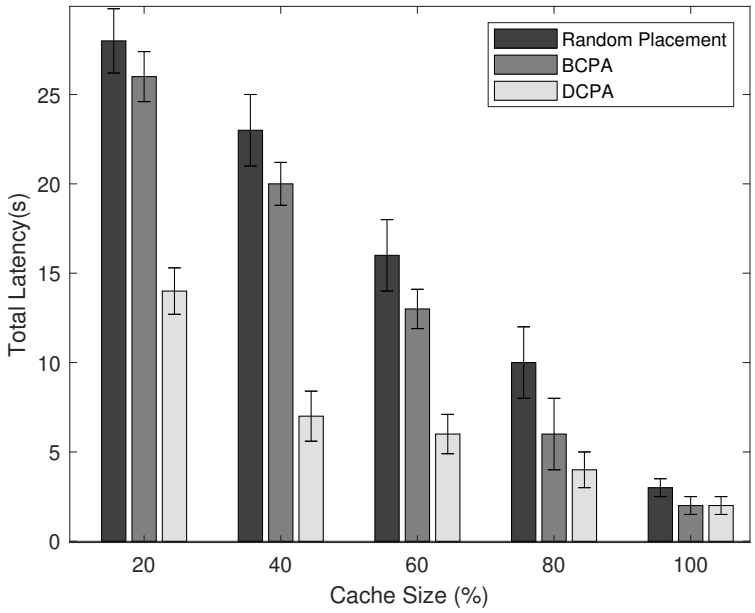


Figure 5.5: Total latency versus cache size.

In Figure 5.5, we analyze the impact of various caching capacity ratios on total latency. We define the caching capacity ratio as the proportion of cached content to all content, and evaluate 95% confidence intervals. As expected, DCPA provides the best delay performance while random being the worst. BCPA is consistently outperformed by DCPA and operates very close to random placement for limited storage capacities. When the cache size is decreased from 100% (everything is cached) to 20%, performance of all methods decreases. Nevertheless, we observe a sharp increase in total latency for BCPA with lower cache sizes, indicating that

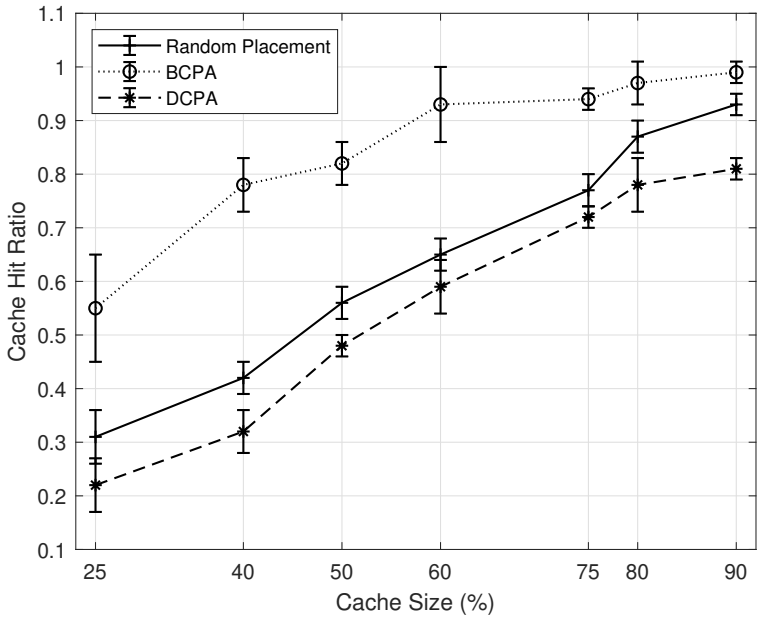


Figure 5.6: Cache hit ratio versus cache size.

BCPA is not able to cache files with potentially more requests due to its batch size limitations and absence of replacement policy. Therefore, we conclude that BCPA is not suitable when caching capacities are limited.

In Figure 5.6, we observe cache hit ratio of the system, i.e, the ratio between the total number of requests and the number of requests satisfied from caches, with various cache sizes. We see that with cache size equal to 90%, for both DCPA and BCPA, almost all the requested content are satisfied from the edge caches indicating a hit ratio of 100%. This is an expected result since with higher cache sizes, the algorithms are able to cache more content. When the cache size is low, the hit ratio gap between BCPA and DCPA is large which can be explained by poor performance of BCPA compared to DCPA.

5.2.5 Impact of File Size

In real life, not all files have the same size. Distinguishing different file sizes may enable us to increase efficiency further. Therefore, we performed experiments by

assigning different sizes to files. We use weighted random distribution where the size of each file is chosen among five different sizes: 50 MB, 100 MB, 200 MB, 500 MB and 1 GB.

Figure 5.7 shows the behavior of algorithms with various average file size. As can be seen from the figure, when the average file size increases total latency increases. This is because we are able to cache less files with larger file size.

When we compare performance of the random algorithm and others, we notice that smaller file size causes the better performance. Being able to cache more files with a lower delay clearly shows that our proposed algorithms work better.

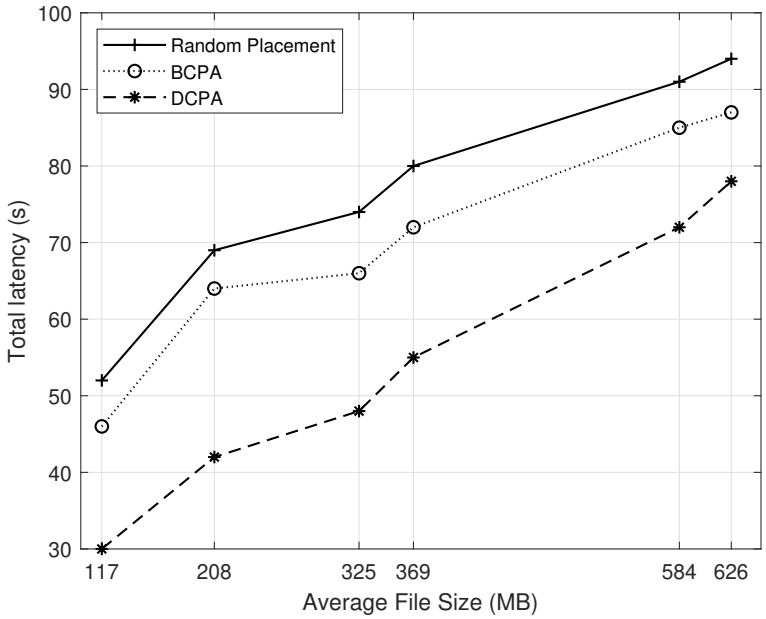


Figure 5.7: Total latency versus average file size.

5.2.6 Impact of Correlated Parameters

Considering the fact that some parameters can also affect each other, we study parameter correlations and their effect on total latency. Doing that, we aim to detect an optimal ratio for correlated parameters whose value will also partially

determine the value of other parameters and maximize the delay performance of proposed algorithms.

For these experiments, if F, U, G, N, Nc values do not change during the experiments, they will be set to 1000, 100, 50, 50, and 2 GB, respectively.

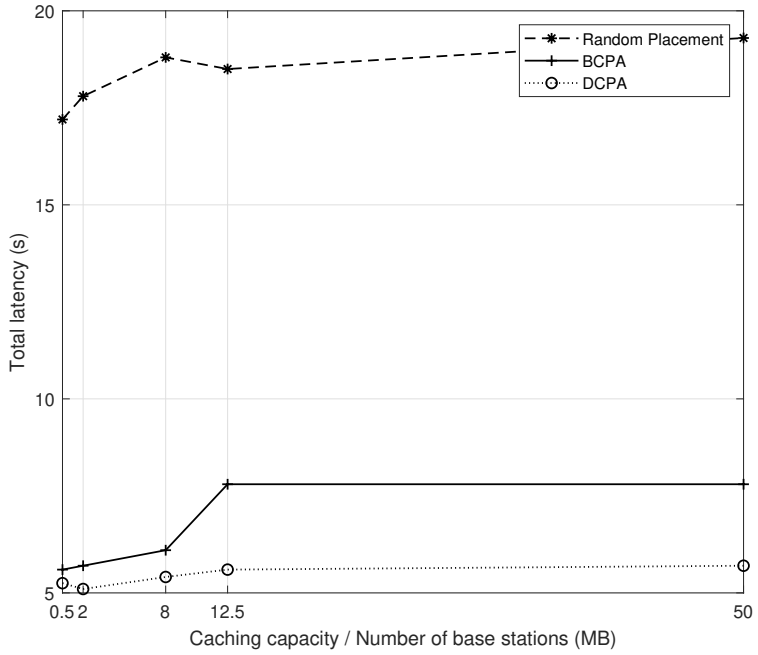


Figure 5.8: Total latency versus $totalCachingCapacity/N$.

As our first correlated parameters, we discuss the effect of total caching capacity and the number of base stations (N). We believe that these two parameters are correlated with each other since total caching capacity directly affects the caching capacity of each base station N. As can be seen from the figure 5.8, having more base stations with limited caching capacity performs better than fewer base stations with greater capacity. This behavior can be explained by the fact that the higher is the N value, the lower is the hop-distances between base stations (D), yielding lower total latency.

Secondly, considering the effect of users per social group on the total number of requests, we observe the effect of number of social groups and users on each other. Figure 5.9 compares the effect of G/U ratio on the total latency. When

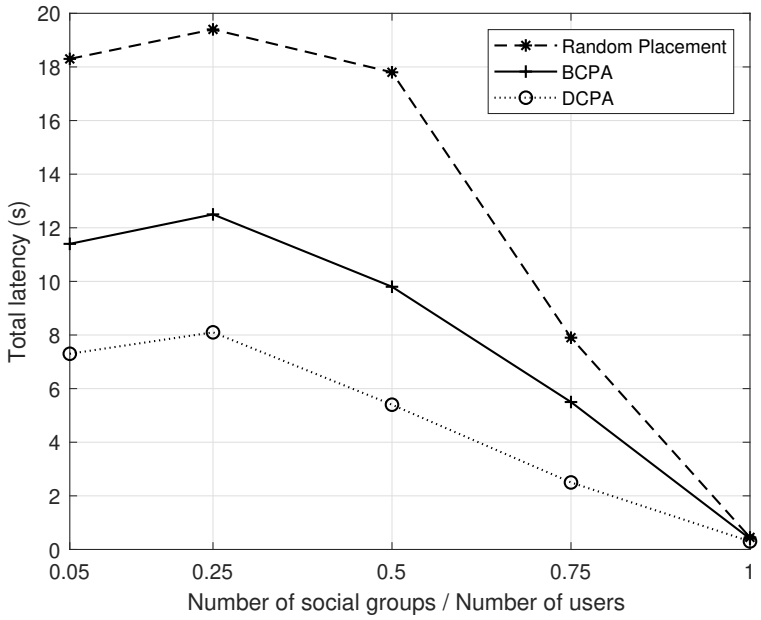


Figure 5.9: Total latency versus G/U.

G Is equal to U, i.e, G/U is 1, the total latency is significantly low for all methods. This corresponds to a case where there are exactly two members in each social group resulting in a significant decrease in the total number of recipient users and hence in total latency. However, we expect these parameters to be not equal. In that case, both BCPA and DCPA perform adequately without a significant improvement in total latency and without changing the system's overall performance significantly.

We also observe the relation between the number of files and users. As one of them determines the total requested contents and the other determines the total requests, we believe there might be a correlation between them. By looking to Figure 5.10, we can say that they are definitely correlated. When we vary F/U ratio from 20 to 100 while keeping the cache size constant, total latency decreases indicating that having more content with less requests leads to higher delay compared to having less content with more requests. It is also important to notice that BCPA operates very close to random placement for all F/U values.

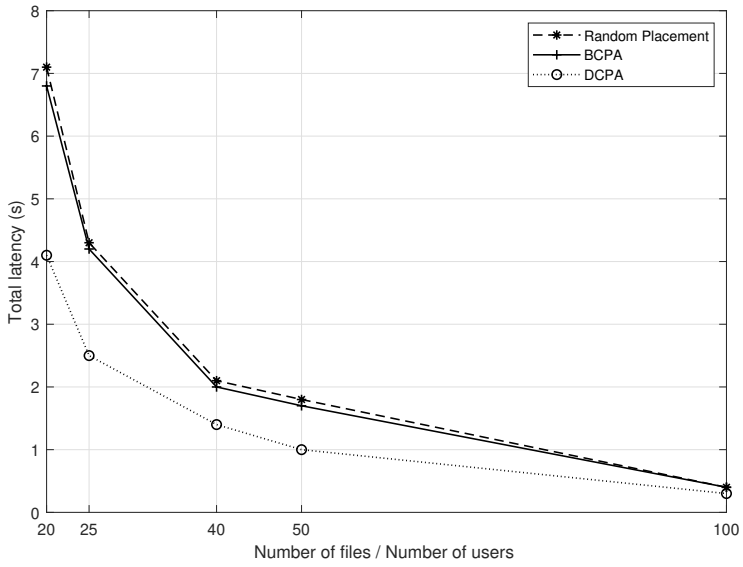


Figure 5.10: Total latency versus F/U.

5.3 Summary

In this chapter, we presented different simulation results and evaluate the performance of our linear model together with our heuristic algorithms. To do that, we compared the performance improvements offered by our methods over random placement, where caching decisions are made randomly.

We first compared two different sorting criteria for BCPA and tried to select the best performing one to be used for the rest of experiments. We observed that sorting files by their total number of recipient groups does not perform well and therefore we excluded that method from the rest of our experiments.

We then analyzed the performance of methods for large parameter values. DCPA provides the best delay performance while random being the worst. BCPA is consistently outperformed by DCPA and operates very close to random placement for most of the experiments. This is because BCPA is not able to cache files with potentially more requests due to batch size limitations and absence of a replacement policy. As DCPA is able to replace existing files if the total latency can be reduced further, it improves the delay performance significantly. Thus, we

concluded that BCPA is not suitable especially for limited caching capacities.

BCPA's efficiency in caching content greatly depends on the batch size and the arrival order of files. DCPA performs very close to the optimal solution.

Chapter 6

Conclusion

In this thesis we addressed the problem of increased data traffic and latency within the mobile social networks. To overcome this problem, we proposed a socially-aware edge caching system model and approach considering user-specified social groups in caching decisions together with storage and transmission capacities of edge servers.

As already mentioned in several studies, social interactions among users dictate the traffic pattern of social network applications. Therefore, identifying users' attitudes toward different content is very important to ensure good user experience in these networks. One of the main contributions of our work is a system where users do not engage with all contents shared by their social groups. Users may have different social groups including their family, friends, acquaintances, neighbors where they are neither obligated to share the same interests nor be attentive to the shared content.

Keeping this in mind and aiming reduced access latency to content, a new approach that prioritizes users' social relationships and their preferences towards files was introduced for solving the social-aware caching optimization problem.

We showed that social-aware cache content placement is an NP-hard problem. Therefore, we proposed a heuristic algorithm, called batch placement caching

algorithm (BCPA), where a set of files are placed in the base stations at a time. The method divides the files into several batches based on their arrival orders and places them accordingly. After conducting extensive simulation experiments, we observed that this method reduces network traffic and latency while receiving content. However, its efficiency in caching content greatly relies on the batch size and the arrival order of files.

We also proposed another heuristic algorithm, called dynamic cache placement algorithm (DCPA). This methods places the files one by one into base stations as they arrive. The algorithm uses a replacement based recursive approach to perform better placement. The main difference of the scheme presented from batch placement is that it continuously checks all the edge caches in order to achieve an optimal solution. This contribution allows us to solve edge caching problem with higher complexity. We also verified this fact, i.e., DCPA being the best performing algorithm, through our simulations. Our simulation results also indicate that although DCPA method improves the system's delay performance significantly, it lowers the computational performance of the servers due to infinite run of algorithm for updating the data and edge caches.

We think that our findings can be important for real-life modelling. However, many different adaptations, tests, and experiments have been left for the future due to lack of time. Future work concerns deeper analysis of particular mechanisms, new proposals to try different methods, or simply curiosity. In this respect, the following ideas could be tested.

To reduce the latency further, caching the same content in different BSs by enabling collaboration between edge servers can be performed. This approach is called collaborative caching where total latency can be lowered in the expense of storage capabilities. Currently, there are some studies where MEC servers are collaborating for executing computation tasks and data caching [46]. This method decreases the delay and makes backhaul links to not suffering from huge data exchange between users and remote clouds.

Coded caching approach may also substantially lower network latency and load.

The idea behind the coded caching scheme is to design an edge caching mechanism where multi-casting is enabled [39, 40]. In this way, when multiple users request different files, their requests can be satisfied with a single coded transmission. This results in a significantly better performance compared to uncoded schemes. That is, by jointly optimizing both the placement and delivery phases, different demands can satisfied with a single multicast transmission.

To ensure system security, two main approaches can be tested. These approaches are called prevention based, i.e., authentication and encryption and detection based, i.e., intrusion detection [47] approaches. However, both approaches are based on trust relationships between users. Thus, adding a new parameter to our model thay represents the social trust level between users may enhance the security of our system.

Bibliography

[1] GSMA, "The state of mobile internet connectivity report 2020 - mobile for development," 2020. (accessed Dec 1, 2020).

[2] Cisco, "Cisco annual internet report (2018–2023) white paper," 2020. (accessed Dec 15, 2020).

[3] H. Currey, S. Cox, and S. Kemp, "Digital 2020: 3.8 billion people use social media," 2020. (accessed Jan 1, 2021).

[4] YouTube, "Youtube statistics," 2021. (accessed Jan 1, 2021).

[5] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[6] J. Nakazato, M. Nakamura, Y. Tao, G. K. Tran, and K. Sakaguchi, "Benefits of mec in 5g cellular networks from telecom operator's view points," in *IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, 2019.

[7] J. Davis and P. Shih, "State of the edge report 2020," 2020. (accessed Dec 1, 2020).

[8] A. Patrizio, "Idc: Expect 175 zettabytes of data worldwide by 2025," 2018. (accessed Dec 8, 2020).

[9] Communications-Today, "Mobile operators struggling to cope with rise in data consumption," 2020.(accessed Feb 1, 2021).

[10] Bitmovin, "Quality of experience: Where quality means doing it right when no one is watching (yet)," 2020. (accessed Feb 3, 2021).

[11] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *European Telecommunications Standard Institute white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[12] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal, *et al.*, "Mobile-edge computing introductory technical white paper," *White paper, mobile-edge computing (MEC) industry initiative*, vol. 29, pp. 854–864, 2014.

[13] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[14] E. Ahmed and M. H. Rehmani, "Mobile edge computing: opportunities, solutions, and challenges," *Future Generation Computer Systems*, 2017.

[15] T. X. Tran and D. Pompili, "Octopus: A cooperative hierarchical caching strategy for cloud radio access networks," in *IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 154–162, 2016.

[16] Y. Cui, W. He, C. Ni, C. Guo, and Z. Liu, "Energy-efficient resource allocation for cache-assisted mobile edge computing," *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, 2017.

[17] X. Li, X. Wang, and V. C. M. Leung, "Weighted network traffic offloading in cache-enabled heterogeneous networks," in *IEEE International Conference on Communications (ICC)*, pp. 1–6, 2016.

[18] 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, 2016.

[19] C. Yen, F. Chien, and M. Chang, "Cooperative online caching in small cell networks with limited cache size and unknown content popularity," in *3rd International Conference on Computer and Communications (ICC)*, pp. 173–177, 2018.

[20] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," *Networked Systems Design and Implementation (NSDI)*, pp. 17–32, 2010.

[21] C. Curino, Y. Zhang, E. Jones, and S. Madden, "Schism: a workload-driven approach to database replication and partitioning," *PVLDB*, vol. 3, pp. 48–57, 2010.

[22] Q. Duong, S. Goel, J. Hofman, and S. Vassilvitskii, "Sharding social networks," pp. 223–232, 2013.

[23] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez, "The little engine(s) that could: Scaling online social networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 4, pp. 1162–1175, 2012.

[24] X. Jiang, T. Zhang, and Z. Zeng, "Content clustering and popularity prediction based caching strategy in content centric networking," in *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, 2017.

[25] M. Gregori, J. Gómez-Vilardebó, J. Matamoros, and D. Gündüz, "Wireless content caching for small cell and d2d networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1222–1234, 2016.

[26] A. M. Vegni and V. Loscrí, "A survey on vehicular social networks," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2397–2419, 2015.

[27] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An analysis of facebook photo caching," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, p. 167–181, 2013.

[28] M. Peng and K. Zhang, "Recent advances in fog radio access networks: Performance analysis and radio resource allocation," *IEEE Access*, vol. 4, pp. 5003–5009, 2016.

[29] L. Braun, A. Klein, G. Carle, H. Reiser, and J. Eisl, "Analyzing caching benefits for youtube traffic in edge networks — a measurement-based evaluation," in *IEEE Network Operations and Management Symposium*, pp. 311–318, 2012.

[30] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Watch global, cache local: Youtube network traffic at a campus network - measurements and implications," *Proceedings of The International Society for Optical Engineering (SPIE)*, vol. 6818, 2008.

[31] Z. Wang, L. Sun, X. Chen, W. Zhu, J. Liu, M. Chen, and S. Yang, "Propagation-based social-aware replication for social video contents," in *Proceedings of the 20th ACM International Conference on Multimedia*, MM '12, p. 29–38, 2012.

[32] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa, "Cooperative caching for efficient data access in disruption tolerant networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 3, pp. 611–625, 2014.

[33] X. Wang, S. Leng, and K. Yang, "Social-aware edge caching in fog radio access networks," *IEEE Access*, vol. 5, pp. 8492–8501, 2017.

[34] H. Chen and W. Lou, "Gar: Group aware cooperative routing protocol for resource-constraint opportunistic networks," *Computer Communications*, vol. 48, pp. 20 – 29, 2014.

[35] Z. Su, Q. Xu, F. Hou, Q. Yang, and Q. Qi, "Edge caching for layered video contents in mobile social networks," *IEEE Transactions on Multimedia*, vol. 19, no. 10, pp. 2210–2221, 2017.

[36] D. Wu, B. Liu, Q. Yang, and R. Wang, "Social-aware cooperative caching mechanism in mobile social networks," *Journal of Network and Computer Applications*, vol. 149, p. 102457, 2020.

[37] W. Luzhi, S. hu, M. Li, and J. Zhou, "An exact algorithm for minimum vertex cover problem," *Mathematics*, vol. 7, p. 603, 2019.

[38] R. Karp, "Reducibility among combinatorial problems," vol. 40, pp. 85–103, 1972.

[39] U. Niesen and M. Maddah-Ali, "Coded caching with nonuniform demands," *Proceedings - IEEE INFOCOM*, 2013.

[40] M. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Transactions on Information Theory*, vol. 60, 2012.

[41] R. Jain, D. M. Chiu, and H. WR, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *Computing Research Repository (CoRR)*, 1998.

[42] Y. Le, L. Ma, W. Cheng, X. Cheng, and B. Chen, "A time fairness-based mac algorithm for throughput maximization in 802.11 networks," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 19–31, 2015.

[43] D. Wei, K. Zhu, and X. Wang, "Fairness-aware cooperative caching scheme for mobile social networks," in *2014 IEEE International Conference on Communications (ICC)*, pp. 2484–2489, 2014.

[44] M. Dianati, X. Shen, and S. Naik, "A new fairness index for radio resource allocation in wireless networks," in *IEEE Wireless Communications and Networking Conference, 2005*, vol. 2, pp. 712–717 Vol. 2, 2005.

[45] N. Kumar and S. Sharma, "Survey analysis on the usage and impact of whatsapp messenger," *Global Journal of Enterprise Information System*, vol. 8, p. 52, 2017.

[46] A. Ndikumana, S. Ullah, T. LeAnh, N. H. Tran, and C. S. Hong, "Collaborative cache allocation and computation offloading in mobile edge computing," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 366–369, 2017.

[47] Y. He, F. R. Yu, N. Zhao, and H. Yin, "Secure social networks in 5g systems with mobile edge computing, caching, and device-to-device communications," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 103–109, 2018.