Fall 2016

Online algorithms for content caching: an economic perspective

Ammar Gharaibeh
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

ONLINE ALGORITHMS FOR CONTENT CACHING: AN ECONOMIC PERSPECTIVE

by
Ammar Gharaibeh

Content Caching at intermediate nodes, such that future requests can be served without going back to the origin of the content, is an effective way to optimize the operations of computer networks. Therefore, content caching reduces the delivery delay and improves the users' Quality of Experience (QoE). The current literature either proposes offline algorithms that have complete knowledge of the request profile a priori, or proposes heuristics without provable performance. In this dissertation, online algorithms are presented for content caching in three different network settings: the current Internet Network, collaborative multi-cell coordinated network, and future Content Centric Networks (CCN). Due to the difficulty of obtaining a prior knowledge of contents' popularities in real scenarios, an algorithm has to make a decision whether to cache a content or not when a request for the content is made, and without the knowledge of any future requests. The performance of the online algorithms is measured through a competitive ratio analysis, comparing the performance of the online algorithm to that of an omniscient optimal offline algorithm. Through theoretical analyses, it is shown that the proposed online algorithms achieve either the optimal or close to the optimal competitive ratio. Moreover, the algorithms have low complexity and can be implemented in a distributed way. The theoretical analyses are complemented with simulation-based experiments, and it is shown that the online algorithms have better performance compared to the state of the art caching schemes.

ONLINE ALGORITHMS FOR CONTENT CACHING: AN ECONOMIC PERSPECTIVE

by
Ammar Gharaibeh

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Engineering

Helen and John C. Hartmann Department of
Electrical and Computer Engineering

January 2017

APPROVAL PAGE

ONLINE ALGORITHMS FOR CONTENT CACHING: AN ECONOMIC PERSPECTIVE

Ammar Gharaibeh

Dr. Abdallah Khreishah, Dissertation Advisor — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

Dr. Nirwan Ansari, Committee Member — Date
Distinguished Professor of Electrical and Computer Engineering, NJIT

Dr. Roberto Rojas-Cessa, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Mengchu Zhou, Committee Member — Date
Distinguished Professor of Electrical and Computer Engineering, NJIT

Dr. Dutta Ashutosh, Committee Member — Date
Lead Member of Technical Staff, Chief Security Office, AT&T Labs

BIOGRAPHICAL SKETCH

Author: Ammar Gharaibeh

Degree: Doctor of Philosophy

Date: January 2017

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2017
- Master of Science in Computer Engineering,
  Texas A&M University, College Station, TX, 2009
- Bachelor of Science in Electrical Engineering,
  Jordan University of Science and Technology, Irbid, Jordan, 2006

Major: Computer Engineering

Presentations and Publications:

A. Gharaibeh, A. Khreishah, I. Khalil and J. Wu, "Distributed Online Enroute Caching," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3455-3468, 2016.

A. Gharaibeh, A. Khreishah and I. Khalil, "An $\mathcal{O}(1)$-Competitive Online Caching Algorithm for Content Centric Networks," *IEEE International Conference on Computer Communications (INFOCOM)*, San Francisco, CA, 2016.

A. Khreishah, J. Chakareski, A. Gharaibeh, I. Khalil and A. Diabat, "Towards Efficient Operation of Internet Data Center Networks: Joint Data Placement and Flow Control for Cost Optimization," *Elsevier Simulation Modelling Practice and Theory*, vol. 64, pp. 83-98, 2016.

A. Khreishah, H. Bany Salameh, I. Khalil and A. Gharaibeh , "Renewable Energy-Aware Joint Caching and Routing for Green Communication Networks," *IEEE Systems Journal*, DOI: 10.1109/JSYST.2016.2530695, 2016.

A. Khreishah, J. Chakareski and A. Gharaibeh, "Joint Caching, Routing, and Channel Assignment for Collaborative Small-Cell Cellular Networks," *IEEE JSAC Series on Video Distribution over Future Internet*, 2016.

A. Gharaibeh, A. Khreishah, B. Ji and M. Ayyash, "A Provably Efficient Online Collaborative Caching for Multicell-Coordinated Systems," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1863 - 1876, 2016.

A. Khreishah, J. Chakareski, I. Khalil, A. Gharaibeh and Y. Jararweh, "Joint Data Placement and Flow Control for Cost-Efficient Data Center Networks," *International Conference on Information and Communication Systems*, Amman, Jordan, 2015.

A. Gharaibeh, A. Khreishah, I. Khalil and J. Wu, "Asymptotically-Optimal Incentive-Based EnRoute Caching Scheme," *IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, Philadelphia, PA, 2014.

A. Khreishah, I. Khalil, A. Gharaibeh, H. Bany Salameh and R. Alasem, "Joint Caching and Routing for Greening Computer Networks with Renewable Energy Sources," *International Conference on Future Internet of Things and Cloud*, Barcelona, 2014.

To my parents Mahmoud and Samiah, my siblings Bashar and Aseel, my wife Yasmeen and my kids Majd and Karam who always believed in me, encouraged me, and supported me through everything.

ACKNOWLEDGMENT

First of all, I thank Allah for all his blessings. I thank Allah for the gift of patience He gave me to do this research and I thank Him for the opportunity to meet all kinds of people through the research process.

Second of all, I thank my advisor, Dr. Abdallah Khreishah for his guidance and patience in the last three years. It is been a pleasure working under such a great professor.

I also want to thank Dr. Nirwan Ansari, Dr. Roberto Rojas-Cessa, Dr. Mengchu Zhou and Dr. Dutta Ashutosh for honoring me as members of my dissertation committee. I also thank them for their feedback on this research.

I would also like to thank all the professors who I have collaborated with to realize this research. I would especially like to thank Dr. Issa Khalil, Dr. Bo Ji, Dr. Moussa Ayyash, and Dr. Jacob Chakareski for the useful discussions I had with them. These discussions opened my eyes on some important things both in life and this research.

Finally, I thank my parents, my brother, my sister, my wife and my sons for their support and faith in me. I am thankful for their prayers to Allah to help me throughout this research.

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

Recently, content retrieval has dominated the Internet traffic. Services like Video on Demand account for 54% of the total Internet traffic, and the ratio is expected to grow to 71% by the end of 2019 [1]. Serving all requests for contents from the origin server leads to server overload and high delays for the users. Content Delivery Network (CDN) uses content replication schemes at dedicated servers to bring the contents closer to the requesting customers. This has the effect of offloading the traffic from the origin servers, reducing content delivery time, and achieving better performance, scalability, and energy efficiency [2, 3]. Akamai, for example, is one of the largest CDNs deployed, delivering around 30% of web traffic through globally-distributed platforms [4]. The problem with CDN is the necessity of *dedicated servers* and that content replication is done *offline*. Content Caching at intermediate nodes is an effective way to optimize the operations of computer networks and to overcome the *dedicated servers* issues, so that future requests can be served without going back to the origin of the content, thus reducing the server overload, the delivery delay, and improving the users Quality of Experience (QoE). To overcome the second issue of *offline* caching, we propose different *online* caching algorithms for different network settings. The difference between offline and online algorithms is explained in the next section.

In this dissertation, we study the problem of content caching under a framework that brings incentives for the nodes to cache the contents. In this framework, Content Providers (CPs) are required to pay charges to the Internet Service Provider (ISP) in

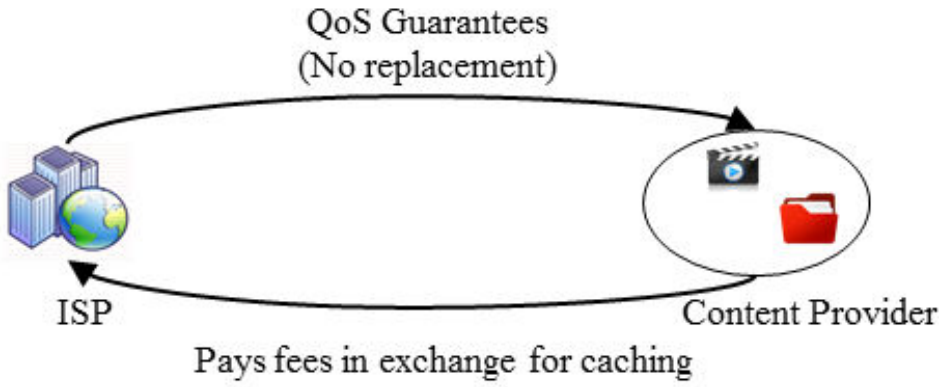

Figure 1.1 Economic framework for caching.

exchange of caching their contents. These charges can include, but not limited to, the price of the storage required by the content providers, the power consumption of the storage devices, among other charges. The definition of these charges is left to the ISP and is beyond the scope of this dissertation. The ISP in return is required to provide some form of Quality of Service (QoS) guarantees by not replacing the contents of the content providers in the future, if the ISP decides to cache their contents. The economic framework is illustrated in Figure 1.1. We propose three different online caching algorithms that take the charges paid by the content providers into consideration in order to achieve their respective objectives of either maximizing the traffic savings (Chapter 3) or minimizing the total cost paid by the content providers (Chapters 4 and 5).

1.1 Definitions

1.1.1 Offline vs. Online Algorithms

The main difference between the offline and the online algorithms is that the offline algorithm has a complete knowledge of the future. In our work, offline means that

the algorithm knows *when*, *where*, and *how many times* a content will be requested. This knowledge leads to the optimal content distribution strategy that maximizes the performance. Conversely, online algorithms do not possess such knowledge. Online algorithms have to make a caching decision for a content based on the available information at the time of the request arrival. Due to this difference, the offline algorithm's performance is typically better than or the same as that of the online algorithm.

In the online version of the problem, the users' requests for contents are revealed one by one. The online algorithm has to make a decision of which content to cache and where to cache it. The algorithm's decisions cannot be changed in the future, and the decisions must be made before the next request is revealed, so the online algorithm works without the knowledge of future users' requests as opposed to the offline algorithm.

1.1.2 Competitive Ratio

To compare the performance of an online algorithm to that of the optimal offline algorithm, we use the concept of competitive ratio. We define the competitive ratio as the worst-case ratio of the performance achieved by the online algorithm to the performance achieved by the optimal offline algorithm. Depending whether the problem is a maximization problem (Chapter 3) or a minimization problem (Chapters 4 and 5), the definition of the competitive ratio is slightly different. Let the performance of the online algorithm be denoted by $\mathcal{P}_{on}$, and the performance of the offline algorithm

be denoted by $\mathcal{P}_{off}$, then for a maximization problem, the competitive ratio is:

$$\sup_{\substack{all\ input \\ sequences}} \frac{\mathcal{P}_{off}}{\mathcal{P}_{on}}.$$

while for a minimization problem, the competitive ratio is defined as:

$$\sup_{\substack{all\ input \\ sequences}} \frac{\mathcal{P}_{on}}{\mathcal{P}_{off}}.$$

As the ratio gets closer to 1, the online performance gets closer to the offline performance, and the better the online algorithm's performance is.

1.2 Dissertation's Outline

We start by presenting an online caching algorithm in Chapter 3 for the current Internet network. In the current Internet network, a request for a content is forwarded along a single path to the content origin server. Our proposed online Cost-Reward Caching (CRC) algorithm decides which nodes along this path are to cache the content with the objective of maximizing the traffic savings. The caching decision is based on a comparison between the value of a cost function that is exponentially proportional to the relative load of the caching node and the expected traffic savings achieved if the content is to be cached. Although estimating the traffic savings requires an estimation of the contents' popularities, we show that knowing the contents' popularities alone is not enough to obtain an optimal solution. The order in which the requests for contents arrive makes a big difference. Our algorithm is easily implemented in the current Internet networks in a distributed way, and we show analytically that our algorithm achieves the optimal competitive ratio in the asymptotic sense. We also propose several extensions

to the CRC algorithm. In the first extension, the CRC algorithm is modified in order to consider content replacement. In the second extension, the objective is to minimize the energy consumed by the network due to caching and transferring contents. The third extension is a combination of the previous two extensions. Our simulation results show that the CRC algorithm and its extensions outperform heuristic methods that are deployed in the current networks such as Cache All and Random caching.

The difficulty of obtaining a prior knowledge of contents' popularities in real scenarios motivated us to design online algorithms that do not require a prior knowledge of the contents popularities. In Chapter 4, we present the Online Collaborative Caching (OCC) algorithm for the Multi-cell coordinated networks that does not require a prior knowledge of the contents' popularities. The network consists of the Mobility Management Entity (MME) of the cellular network which acts as a centralized controller, and multiple base stations that can collaborate with each other to satisfy requests for contents made by the users of the cellular network. The execution of the algorithm is done by the MME. The algorithm works on a per-request basis, where upon the arrival of a request for a content, the MME updates the value of a potential function of each base station, decides to cache the content if the value of the potential function exceeds the caching cost, and relays its decision to the base stations. The potential function is a measure of how beneficial it is to cache at a base station. Our algorithm decides which base stations are to cache the contents and from which base station a user's request is satisfied with the objective of minimizing the total cost paid by the content providers. We also show that our algorithm achieves a competitive ratio that is close to the optimal. Through extensive simulations, we show that the collaborative

caching schemes provide higher savings than the non-collaborative caching scheme, which means that applying the simple online algorithm is better than solving the non-collaborative optimization problem.

Having a centralized controller may introduce some operational issues such as overloading the controller or security concerns. This motivated us to design an algorithm that can be implemented in a distributed way in Chapter 5, where we consider content caching in Content Centric Networks (CCN). Routing in CCN is different from routing in the current networks in that the routing in CCN is based on the content's name instead of the IP address of the content's source, which means that a content can be retrieved from nodes that are not on the path to the content's origin server. For the CCN network, we present Online algorithm for Caching in CCN (OC^3N) that decides where to cache a content and from which node a user's request is satisfied in order to minimize the total cost paid by the content providers. The algorithm does not require a prior knowledge of the contents' popularities and can be implemented in a distributed way. Upon the arrival of a request for a content at a node, that node becomes responsible of executing the algorithm. This is done by exchanging messages with other nodes within the vicinity in order to collect the necessary information to execute the algorithm. The node calculates the value of a potential function of each of the participating nodes, and depending on the caching costs, decides whether to cache the content at a new location or not. The node then relays the caching decision and other information to the participating nodes in order for these nodes to update their own potential function. Through theoretical analysis, we show that OC^3N achieves a constant competitive ratio when the caching cost changes periodically, and the content

items are evicted at the end of the period. For the cases where the caching cost does not change, we propose a heuristic based on OC^3N where the content items are cached for a fixed time. We complement the theoretical analysis with simulations and show that our algorithm can achieve up to 65% less cost than widely used caching schemes in CCN such as Leave Copy Down (LCD) and Leave Copy Everywhere (LCE).

Table 1.1 presents a summary of the proposed algorithms.

The dissertation is finally concluded in Chapter 6.

Table 1.1 Summary of The Proposed Online Algorithms

Algorithm	**Network Type**	**Competitive Ratio**	**Lower Bound**	**Centralized/ Distributed**
CRC Algorithm (Chapter 3)	Current Internet Network	$\mathcal{O}(\log(N))$ [1]	$\Omega(\log(N))$	Distributed
OCC Algorithm (Chapter 4)	Multi-cell Coordinated Network	$\mathcal{O}(\log(n))$ [2]	$\Omega(\frac{\log(n)}{\log\log(n)})$	Centralized
OC^3N Algorithm (Chapter 5)	Content Centric Network	$\mathcal{O}(1)$	–	Distributed

[1] N here represents the number of nodes in the network
[2] n here represents the number of requests for contents

CHAPTER 2

RELATED WORK

Content caching has been extensively studied in the literature from different perspectives and several techniques for content caching have been proposed. Caching from an economical perspective has been studied, but the studies are limited to offline solutions [5–12], or do not provide provable guarantees for the proposed online solution [13]. Looking at caching from other perspectives, either offline approximation algorithms are proposed [14, 15], online algorithms without provable guarantees are proposed [16–22], or online algorithms with provable performance guarantees [23]. This dissertation is different from all the above-mentioned work is that we study caching from an economic perspective and propose online algorithms with provable performance guarantees. In the following, we present a brief discerption of the most related works.

Looking from an economic perspective, the works in [5] and [6] consider incentives for nodes to cache. However, they provide high level solutions in an offline fashion. The authors in [6] consider a special case with only 3 ISPs. the work in [7] proposes an optimization problem to minimize the total cost of content retrieval given content items' popularities, cache sizes, and links' capacities. The optimization problem is solved in an offline fashion, where the exact content items' popularities are known. The work of [8] considers maximizing the profit of the clouds provider through caching in an offline fashion. A game theoretic model between the ISP and the CP is proposed in [9] to fairly split the caching cost/profit between the ISP and the CP. The work in [24] provides simulation-based analysis showing that in-network caching

provides cost savings compared to caching in data centers only. The work in [25] proposes an offline optimization problem that requires the knowledge of contents' popularities in order to maximize the hit ratio given a price for caching.

In [10], the authors study collaborative caching among small base stations deployed in a single macro-cell. The objective of the study is to minimize the cellular network operational cost, given the cache size at each small base station and the bandwidth of the backhaul links. The authors also propose a combined offline algorithm and Least Frequently Used (LFU) replacement policy for in-network cache management, and prove that the ratio of the performance of the offline algorithm to the optimal offline algorithm is within a factor of β, which is linear in terms of the product of the number of potential collaborators, the number of requests for each user, and the number of cached contents at each small base station. Our work is different in that we propose an online caching algorithm that achieves a better performance ratio when compared to the optimal offline algorithm. The authors in [11] propose an offline caching scheme to maximize the reward gained by the cellular network operator when the cache of each base station is limited. An optimal offline greedy algorithm is proposed in [12, 26] to minimize the cost of the ISP. However, the authors only consider link costs and do not consider caching costs. The work in [27] studies content pre-staging (i.e., offline caching) through a Stackelberg game model between the mobile wireless operator, the content provider, and the mobile users, where the content providers are charged for using the mobile users devices as storage, and the mobile users are paid for the use of their devices. The work in [28] also proposes a game theoretic approach between the ISP and the CP with the objective of minimizing

the ISP's operational costs. The game is performed in an offline fashion where the contents' popularities are known and the cost of accessing a content is set depending on its popularity. All the above-mentioned studies consider caching in an offline fashion.

The work in [13] proposes online algorithms for minimizing caching and traffic costs. The work in [29] proposes an online algorithm for video caching in cellular networks, where the decision are: 1) which video to cache and 2) at which bit rate (in order to enhance Adaptive BitRate (ABR) streaming technology) to minimize the network backhaul cost. In all of these studies, the authors propose online algorithms for caching. However, the authors do not provide any analysis regarding the performance guarantees of their algorithms.

The most related work to this dissertation is the work in [30]. The authors study caching from an economic perspective under a stochastic framework and propose three schemes for contract negotiation between the ISP and the CP. Depending on which scheme is deployed, the negotiation happens when either a cache miss or a cache hit occurs. All three schemes aim to negotiate the period of time the content is to be cached. upon a cache miss (or a cache hit), the CP negotiates (or renegotiates) its contract with the ISP to determine how long to cache the requested content with the objective of maximizing the profit of the CP. Under this stochastic framework, the authors characterize the optimal strategy (in terms of how long to cache the content) for the CP to choose based on the chosen scheme and the pricing policy set by the ISP. However, the choice of the optimal strategy depends on having the knowledge of the distribution of the inter-arrival time of the requests. Our work is different in that we propose online algorithms that do not require such knowledge.

Looking from other perspectives (i.e., other than economic perspective), the work in [14] presents an offline solution through dynamic programming for content placement for en-route caching, while the work in [15] provides approximation algorithms to minimize the number of requests served by the main base station in a cellular network enhanced with Small Base Stations (SBS). These works provide offline solutions to the caching problem.

As for an online point of view, the work in [16] presents *Always Cache*, where a node caches every new piece of content under the constraint of cache capacity. Most Popular Caching caches a content at neighboring nodes when the number of requests exceeds some threshold [17]. The work of [18] presents an online solution but with no efficiency or optimality proofs. The authors in [19] study caching in CCN by formulating an optimization problem with the objective of minimizing the inter-ISP traffic or the average access latency, and also proposes an online algorithm that caches the content items on the en-route path. The work in [20] improves upon the work in [17] by always caching new content items when the cache is underutilized. When the cache is full and the counter of a content reaches a certain threshold, the content is cached using LRU replacement policy. Otherwise, the content will not be cached. The work in [21,22] provides an online caching scheme to minimize the total energy consumption of CCN. In all these works, the authors do not provide any analysis regarding the performance guarantees of the proposed algorithms. In [23], caching is studied in a network where all users are connected to a single server via a shared backhaul link. An online algorithm with provable performance guarantees is proposed that minimizes

the traffic sent over the shared link. The work in [31] proposes an online algorithm for video caching with the objective of maximizing the hit ratio.

Different from the above-mentioned works, other studies have investigated caching in an offline fashion to achieve different objectives. The authors in [32] provide a push-pull model to optimize the joint latency-traffic problem by deciding which contents to push (cache) on intermediate nodes, and which contents to pull (retrieve) from the origin server. *ProbCache* aims to reduce the cache redundancy by caching contents at nodes that are close to the destination [33]. A cooperative approach in [34] leads to a node's caching decision that depends on its estimate of what neighboring nodes have in their cache. A collaborative caching mechanism in [35] maximizes cache cooperation through dynamic request routing. In [36], nodes try to grasp an idea of other nodes' caching policies through requests coming from those nodes.

The objective of the work in [37] is to minimize the delay through content popularity estimation in a single base station, while [38] uses caching helpers to achieve the same objective. The authors in [39–41] consider hierarchical caching in cellular backhaul networks, while [42, 43] take an information-theoretic approach to study hierarchical caching.

The authors in [44] propose a proactive offline caching scheme, and develop a heuristic algorithm to minimize the content access delay of all users. The authors assume that content popularity is the same across different base stations. The work in [45] studies collaborative caching with the objective of minimizing either the inter ISP traffic, the intra ISP traffic, or the overall user delays, using a proactive offline caching scheme and a heuristic algorithm.

The work in [46] provides an optimal caching strategy to minimize the average latency experienced by end users in a CCN when the nodes either coordinate or do not coordinate with each other to reach an optimal caching decision. The work in [47] proposes an energy consumption model for CCN and formulates an optimization problem to minimize the total power consumption of CCN given the content items' popularities and the cache sizes. The work in [48] analyzes the impact of caching on the energy efficiency of cellular networks. All of the above-mentioned studies require the exact knowledge of content items' popularities, which leads to offline solutions, in which the content items are cached in the network before the content items are requested.

Table 2.1 provides a summary of the related work.

Table 2.1 Summary of The Related Work

References	**Economic Perspective**	**Online Algorithm**	**Provable Performance Guarantees**
This work, [30]	•	•	•
[13, 29]	•	•	
[10–12, 27, 28]	•		•
[23, 31]		•	•
[5–9, 24–26]	•		
[14, 15]			•
[16–22]		•	
[32–48]			

CHAPTER 3

ONLINE ALGORITHM FOR CACHING IN THE CURRENT INTERNET NETWORK

3.1 Introduction

In this chapter[1], we present an online caching algorithm for the current Internet network. In the current Internet network, a request for a content is forwarded along a single path to the content origin server. Our proposed online algorithm decides which nodes along this path are to cache the content with the objective of maximizing the traffic savings. This chapter provides a provably-optimal online solution for the first time under a setting that brings incentives for the nodes to cache. In order to provide incentives for the nodes to cache, nodes have to charge content providers for caching their contents. Adopting such charging policies forces the caching node to provide quality of service guarantees for content providers by not replacing their contents in the future, if the node decides to cache their contents. Since the number of contents far exceeds the nodes' cache capacities, and assuming that the charging price for every piece of content is the same, then the node has no preference in caching one content over the other, forcing the node to cooperate and apply our policy that achieves asymptotic optimality.

3.1.1 Settings

A network is represented by a graph $G(V, E)$, where each node $i \in V$ has a caching capacity of D_i. If the node does not have caching capability, its caching capacity is set

[1]The work of this chapter has been published in [49]

to 0. Weights can be assigned to each link $e \in E$, but we consider all links to have the same weight. The input consists of a sequence of contents $\beta_1, \beta_2, ..., \beta_m$, the j-th of which is represented by $\beta_j = (S_j, r_j, T_j(\tau))$, where S_j is the source for content β_j, r_j is the size of β_j, and $T_j(\tau)$ is the effective caching duration in which more requests are expected for β_j when a request appears at time slot τ. For simplicity, we assume a slotted time system and that $T_j(\tau)$ is an integer multiple of slots.

For each content, we define the following values:

1. $b_i(j)$: Number of hops on the path from node i to S_j for β_j.
2. $W_i(\tau, j)$: The expected number of requests for β_j to be served from the cache at node i at time slot τ, if all of the caching nodes cache β_j.
3. $t_0(i, j)$: The time when a request for β_j appears at node i.
4. $\mathcal{E}_i(\tau, j)$: The total expected number of requests for β_j to be served from the cache at node i per time slot τ. We assume that $\mathcal{E}_i(\tau, j)$ is fixed $\forall \tau \in \{t_0, \ldots, t_0 + T_j(t_0)\}$.
5. $\tau_0(i, j)$: The time when β_j is cached at node i. For simplicity, we denote this value hereafter by τ_0 since the values of (i, j) can be inferred from the context.
6. $d_i(\tau, j)$: Number of hops from node i to the first node caching β_j along the path to S_j at time τ. We assume that if node i caches β_j at time τ_0, then $d_i(\tau, j) = d_i(\tau_0, j), \forall \tau \in \{\tau_0, \ldots, \tau_0 + T_j(\tau_0)\}$.

Figure 3.1 shows a simple network to illustrate the aforementioned definitions. In this example, we have two contents β_1 and β_2, originally stored on v_1 and v_2, respectively. The triangles in this figure represent the subnetworks containing the set of non-caching nodes connected to the caching node. The values of $W_i(\tau, j)$ represent the expected number of requests for β_j coming from the set of non-caching nodes in the subnetwork connected to node i.

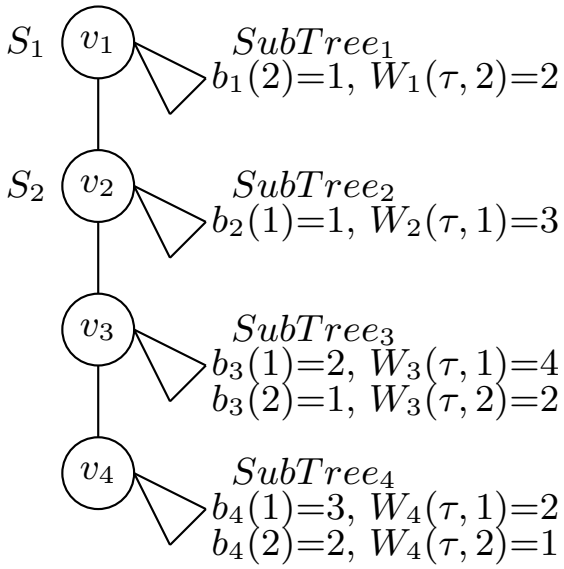

Figure 3.1 Simple caching network.

Before any requests for β_j appears at any node, each node i will send its $W_i(\tau, j)$ to all nodes on the path from node i to the source of β_j, S_j. This process will lead to the calculation of the initial values of $\mathcal{E}_i(\tau, j)$.

For example, in Figure 3.1, before any request for β_1 appears at any node, $\mathcal{E}_3(\tau, 1) = W_3(\tau, 1) + W_4(\tau, 1)$, to a total value of 6. This is because, starting from the initial configuration while investigating the caching of content β_1 on node v_3, all the requests for β_1 coming from the subnetworks connected to v_3 and v_4 will be served from the cache of v_3, if we decide to cache β_1 on v_3. Similarly, $\mathcal{E}_2(\tau, 1) = 9$. Later on, if v_4 decides to cache β_1, then $W_4(\tau, 1)$ will be subtracted from all nodes along the path to S_1, until the first node caching β_1 is reached. This is because none of these nodes will serve the requests for β_1 coming from the subnetwork connected to v_4 after this point. In Sections 3.2 and 3.3.3, we provide details for the dynamic calculation and initialization of $\mathcal{E}_i(\tau, j)$, respectively.

We define the total traffic savings of caching in the time interval [0,t] as:

$$\sum_{\tau=0}^{t}\sum_{i=1}^{n}\sum_{j=1}^{m}\mathcal{E}_i(\tau_0,j)d_i(\tau_0,j)I(a_i(\tau,j)), \tag{3.1}$$

where $I(.)$ is the indicator function and $a_i(\tau,j)$ is the event that β_j exists at node i at time τ. For example, referring to Figure 3.1, caching β_1 on v_3 alone for a single time slot will yield a saving of $\mathcal{E}_3(\tau,1)\times d_3(\tau,1)=(4+2)\times 2=12$.

We define the relative load on a caching node i at time τ when β_j arrives as

$$\lambda_i(\tau,j)=\sum_{\substack{k:k<j\\ k\in Cache_i(\tau)}}\frac{r_k}{D_i},$$

where $k<j$ refers to the indices of all β_k that are in the cache of node i at the time when considering β_j to be cached at node i. We use $k\in Cache_i(\tau)$ to represent the existence of β_k in the cache of node i at time τ.

As we mentioned in Section 3.1, charging content providers for caching their contents will provide the nodes with the necessary incentives to cache. In return, the nodes have to guarantee quality of service for content providers by keeping their content cached for the required time period. We assume that content providers are charged the same to prevent the node from preferring contents with a higher prices. To this end, we consider *non-preemptive* caching to represent our system model, *i.e.*, once β_j is cached at node i, it will stay cached $\forall\tau\in\{\tau_0,\ldots,\tau_0+T_j(\tau_0)\}$ time slots. We elaborate more on $T_j(\tau)$ in Section 3.3.4.

3.1.2 Motivation

We motivate the design of our online algorithm by the following reasoning; knowing the contents' popularities alone does not guarantee an optimal solution. The order in which the contents arrive makes a big difference.

Referring to Figure 3.1, consider the existence of two contents named X and Y, originally located at v_1. Assume that $W_3(\tau, X) = W_4(\tau, X) = 1$, $W_3(\tau, Y) = 1$, and $W_4(\tau, Y) = 10$. Assume that all nodes have enough residual cache capacity for one content except node v_3, which is full and cannot cache any content. Furthermore, assume that X and Y will be both requested twice by v_4 at different time slots. Consider the following two scenarios:

- En-Route Caching: If the first request for content X, followed by the first request for content Y, arrives at v_4, then v_4 will cache the first content X and v_2 will cache content Y, achieving a traffic saving at v_4 for the next pair of requests of $(1 \times 3) + (10 \times 1) = 13$. Later on, if requests for X and Y appear at v_3, then v_3 will get content X from v_1 and content Y from v_2, gaining an additional savings of $(1 \times 0) + (1 \times 1) = 1$.

  Alternatively, if a request for Y is followed by a request for X at v_4, then v_4 will cache the first content Y and v_2 will cache content X, achieving a traffic saving at v_4 for the next pair of requests of $(10 \times 3) + (1 \times 1) = 31$. Later on, if requests for X and Y appear at v_3, then v_3 will get content X from v_2 and content Y from v_1, gaining an additional savings of $(1 \times 1) + (1 \times 0) = 1$. So the online algorithm will achieve an average traffic savings of 23.

  The offline algorithm knows in advance that content Y will be requested and can reject the caching of content X at v_4 and cache it at v_2 to achieve a traffic saving of $(10 \times 3) + (1 \times 1) + (1 \times 1) + (1 \times 0) = 32$.

- Routing to the Closest Caching Node: If the first request for content X, followed by the first request for content Y, arrives at v_4, then v_4 will cache the first content X, and v_2 will cache content Y, achieving a traffic saving at v_4 for the next pair of requests of $(1 \times 3) + (10 \times 1) = 13$. Later on, if requests for X and Y appear at v_3, then v_3 will get content X from v_4 and content Y from v_2, gaining an additional savings of $(1 \times 1) + (1 \times 1) = 2$.

  Alternatively, if a request for Y is followed by a request for X at v_4, then v_4 will cache the first content Y and v_2 will cache content X, achieving a traffic saving

at v_4 for the next pair of requests of $(10 \times 3) + (1 \times 1) = 31$. Later on, if requests for X and Y appear at v_3, then v_3 will get content X from v_2 and content Y from v_4, gaining an additional savings of $(1 \times 1) + (1 \times 1) = 2$. Because of this, the online algorithm will achieve an average traffic savings of 24.

The offline algorithm knows in advance that content Y will be requested and can reject the caching of content X at v_4, and will cache it at v_2 to achieve a traffic saving of $(10 \times 3) + (1 \times 1) + (1 \times 1) + (1 \times 1) = 33$.

The above examples show that the online algorithm cannot guarantee an optimal solution. In fact, we show that there is an upper bound on the savings achieved by the online algorithm when compared to the offline algorithm, and we develop an online algorithm that achieves that bound under realistic settings.

3.2 Algorithm

In this section, we present the Cost-Reward Caching (CRC) algorithm that achieves the optimal competitive ratio, along with some practical issues. We introduce the proof of optimality in the next section.

3.2.1 CRC Algorithm

CRC takes advantage of en-route caching, *i.e.*, a request for a content is forwarded along the path to the content's source, up to the first node that has the content in its cache. The content then will follow the same path back to the requester.

In CCN, when an interest packet for a new content arrives at a node on a certain interface, the node will send the interest packet using all other interfaces. For example, Figure 3.2 shows a single node in CCN, where the numbers represent the interfaces of the node. When a request for β_j arrives at the node through interface number 2, and a match is not found in neither the cache nor the Pending Interest Table (PIT), the node

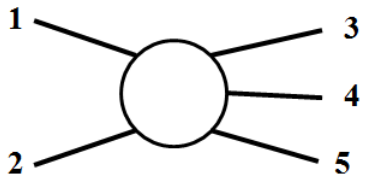

Figure 3.2 A single node in CCN.

will send the request on all interfaces except interface number 2. Our algorithm uses en-route caching, so the new interest packet is only forwarded on the single interface along the path to the content's source.

When a request for a content β_j appears at a node i at time t_0, node i sends a small control message up to the first node caching β_j along the path to the source of the content. Let w be that first node, then node w replies with a message containing r_j and the ID of node w. Every node u in the path from node w to node i stores a copy of the message, computes $d_u(t_0, j)$, and forwards the message to the next node along the path to node i. When Node i receives the message, it makes a caching decision according to Algorithm 2. If node i decides to cache β_j, it initializes a header field in the request packet to the value of $\mathcal{E}_i(\tau, j)$. If node i decides not to cache, it initializes the header field to 0.

The request packet is then forwarded to the parent node z. The parent first subtracts the value stored in the header field from its own value of $\mathcal{E}_z(\tau, j)$. Based on the new value of $\mathcal{E}_z(\tau, j)$, if node z decides to cache β_j, it adds its $\mathcal{E}_z(\tau, j)$ to the value in the header field. Otherwise, node z adds 0. The request packet is then forwarded to node z's parent, and the whole process is repeated until the request reaches the first node that has the content in its cache. The content then will follow the same path back

to the requester, and every node in the path that decided to cache the content will store a copy in its cache. We describe the operation of our algorithm in Algorithm 1.

Algorithm 1 En-Route Caching

A request for β_j appears at node i at time t_0.
$header = 0$
if $\beta_j \in Cache_i(t_0)$ **then**
    Reply back with β_j
else
    Send a control message to retrieve r_j, $d_i(t_0, j)$
    $w \leftarrow$ first node on the path to S_j, where $\beta_j \in Cache_w(t_0)$
    Node w replies with r_j and ID
    $\forall u \in Path(w, i)$, store $r_j, d_u(t_0, j)$
    for $u_k \in Path(i, w), k = 1 : Length(Path(i, w))$ **do**
        $\mathcal{E}_{u_k}(t_0, j) = \mathcal{E}_{u_k}(t_0, j) - header$
        Run *Cost-Reward Caching* algorithm
        if Caching Decision = TRUE **then**
            $header = header + \mathcal{E}_{u_k}(t_0, j)$

For example, Figure 3.3 shows a simple network where a content β_1 is originally stored at S_1. We removed the triangles representing the set of non-caching nodes for the sake of clarity. If a request for β_1 appears at v_0, node v_0 will send a control message up to the first node caching β_1, which is S_1, and retrieves the values of r_1 and $d_0(t_0, 1) = 1$. Based on these values, if v_0 decides to cache β_1, it will send the request for β_1 to its parent, which is S_1, with the header field initialized to $\mathcal{E}_0(t_0, 1) = 14$. Node S_1 will simply reply with a data packet containing β_1, and v_0 will cache β_1. Later on, if another request for β_1 appears at v_5 while β_1 is still cached at v_0, node v_5 will send a control message up to the first node caching β_1, which is v_0. Node v_0 sends a message containing the values of r_1 and its ID to node v_2. Node v_2 will store the value of r_1, sets $d_2(t_0, j) = 1$, and forwards the message to v_5. Node v_5 in turn will store the value of r_1 and set $d_5(t_0, j) = 2$. Based on these values, if v_5 decides to cache β_1 it will send the request for β_1 to its parent, which is v_2, with a header field initialized to $\mathcal{E}_5(\tau, 1) = 2$.

When the request reaches v_2, it will first subtract the value in the header field from its own $\mathcal{E}_2(\tau, 1)$, so the new value of $\mathcal{E}_2(\tau, 1)$ is $\mathcal{E}_2(\tau, 1) = \mathcal{E}_2(\tau, 1) - header = 4 - 2 = 2$. The reason that node v_2 has to subtract the header field from its own $\mathcal{E}_2(\tau, 1)$ is because the requests for β_1 coming from the subnetwork connected to node v_5 will not be served from the cache of node v_2 since v_5 decided to cache β_1. Based on these values, if v_2 decides *not* to cache β_1, it will add 0 to the header field and forward the request to its parent v_0. Node v_0 will simply reply with a data packet containing β_1, and only v_5 will cache β_1.

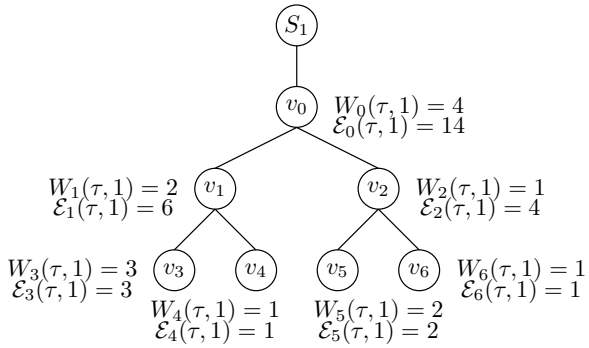

Figure 3.3 Simple caching network 2.

The core idea of the Cost-Reward Caching algorithm is to assign an exponential cost function for each node in terms of the node's relative load. If the cost of caching a content is less than the traffic savings achieved by caching the content, the algorithm decides to cache. The choice of an exponential cost function guarantees that the node's capacity constraints are not violated. We show that in the next section.

We define the cost of caching at a node i at time τ as:

$$C_i(\tau, j) = D_i(\mu^{\lambda_i(\tau,j)} - 1),$$

where μ is a constant defined in Section 3.4. The algorithm for Cost-Reward Caching is presented in Algorithm 2.

Algorithm 2 Cost-Reward Caching (CRC)

1: New request for β_j arriving at node i at time t_0
2: $\forall \tau \in \{t_0, \ldots, t_0 + T_j(t_0)\}$, Compute $\lambda_i(\tau, j), C_i(\tau, j)$
3: **if** $\sum_{\tau=t_0}^{t_0+T_j(t_0)} \mathcal{E}_i(\tau, j) d_i(t_0, j) \geq \sum_{\tau=t_0}^{t_0+T_j(t_0)} \frac{r_j}{D_i} C_i(\tau, j)$ **then**
4: Cache β_j on node i
5: $\tau_0(i, j) = t_0(i, j)$
6: $\forall \tau \in \{t_0, \ldots, t_0 + T_j(t_0)\}, \lambda_i(\tau, j+1) = \lambda_i(\tau, j) + \frac{r_j}{D_i}$
7: **else**
8: Do not cache

In the algorithm, when new content that is *not currently cached by node* i arrives at time t_0, node i computes the relative load ($\lambda_i(\tau, j)$) and the cost ($C_i(\tau, j)$) for every $\tau \in \{t_0, \ldots, t_0 + T_j(\tau)\}$. This is because a currently cached content may be flushed before $t_0 + T_j(t_0)$, thus the relative load and the cost should be adjusted for each time slot thereafter.

For example, Figure 3.4 shows the relative load at a node for the next 10 time slots starting from t_0, which is the arrival time of a new content β_4. The node has three cached contents, β_1, β_2, and β_3 that are going to be flushed at times $\tau_1 = t_0 + 3$, $\tau_2 = t_0 + 9$, and $\tau_3 = t_0 + 7$, respectively. When a β_4 arrives at this node at $\tau = t_0$ with $T_4(t_0) = 10$, the cost calculation should include three cached contents for 3 time slots, two cached contents for 4 time slots, one cached content for 2 time slots, and 0 cached content for 1 time slot. If the total savings for caching β_4 is greater than the aggregated cost, then β_4 will be cached on node i, and the relative load is updated to include the effect of β_4.

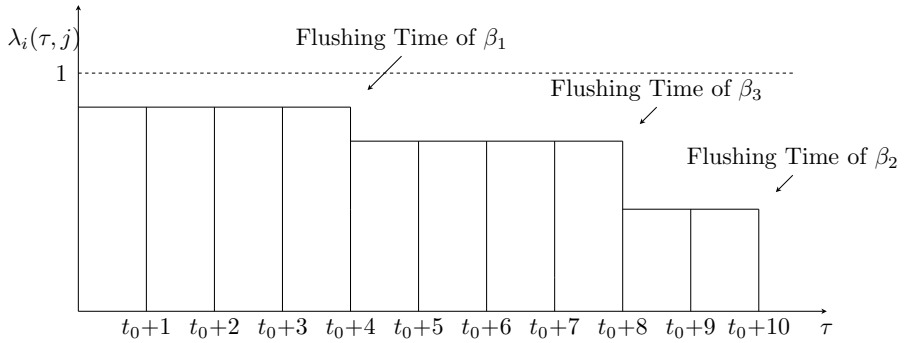

Figure 3.4 Relative load calculation example. The figure shows the state of the cache in one node when it considers a new content β_4 for caching at time t_0 and $T_4(t_0) = 10$. We have three contents, β_1, β_2, and β_3, that are to be flushed at times $\tau_1 = t_0 + 3$, $\tau_2 = t_0 + 9$, and $\tau_3 = t_0 + 7$, respectively.

3.3 Practical Issues

So far, we developed a fully distributed algorithm that achieves asymptotic optimality in terms of traffic savings under some realistic assumptions. Before providing the optimality proof, we discuss in this section the practical issues that make the algorithm easy to implement. The major issues in our algorithm include providing incentives for the caching nodes and QoS guarantees for the content providers, the adoption of en-route caching, calculating the popularity expectation of each content, and updating the effective caching duration.

3.3.1 Providing Incentives and QoS Guarantees

In this work, the QoS measure is to guarantee the existence of the content in the cache for a certain period of time, so the content will be delivered quickly. In other words, once a caching node decides to cache a certain content, the content will not be replaced during the effective caching time of the content. Providing such a guarantee along with adopting an equal pay charging policy for all contents will provide the caching nodes

with the necessary incentive to cache. Figure 3.5 shows the interaction between the ISP and the content provider.

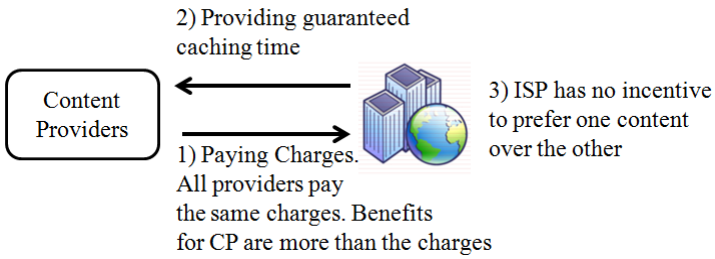

Figure 3.5 Interaction between ISP and content provider.

We assume that the caching nodes should adopt charging policies, where every content provider is charged the same. This will prevent the caching node from preferring one content over the other. Moreover, such charging policies will enforce the caching nodes to cooperate and apply our CRC algorithm

3.3.2 En-Route Caching

In en-route caching, a request for β_j will be sent to the parent along the traditional path to the content's source, until the request reaches the first node caching the content or the content's source. The adoption of this en-route caching reduces the amount of broadcasted *Interest* packets as opposed to the currently deployed schemes in CCN, where the interest packets are broadcasted to all neighbors. Moreover, using en-route caching prevents the reception of multiple copies of the requested content as opposed to CCN. Furthermore, our algorithm can be easily implemented in the current Internet architecture.

3.3.3 Calculating the Initial Content Expectation Values

For each content, we start by building a caching tree rooted at the source of the content. The caching tree is the union of the traditional paths from the source of the content to all other nodes. We calculate the initial expectation value at a caching node for a certain content, when only node S_j holds the j-th content, based on the content's popularity and the number of end nodes in the subnetwork connected to that node. For example, in Figure 3.1, $W_3(\tau, j)$ at node v_3 for content β_j is proportional to the content's popularity and the number of end nodes in the subnetwork connected to node v_3.

Algorithm 3 shows how to calculate $\mathcal{E}_i(\tau, j)$ for each content at each caching node before the appearance of any request at any node. The expectations are calculated in a distributed way, where each node only needs to know the expectation values of its children in the caching tree. In the simulation, we investigate the effect of having error margins in the expectation calculation.

Algorithm 3 Initial Content Popularity Expectation Calculation

1: **for** each content $\beta_j = \{S_j, r_j, T_j(\tau)\}$ **do**
2: $CachingTree(j) \leftarrow$ build the traditional path tree rooted at S_j
3: **for** each caching node $i \in CachingTree(j)$ **do**
4: Calculate $W_i(\tau, j)$
5: Initialize $\mathcal{E}_i(\tau, j) \leftarrow W_i(\tau, j)$
6: **for** each node $z \in Ancestor(i)$ in $CachingTree(j)$ **do**
7: $\mathcal{E}_z(\tau, j) = \mathcal{E}_z(\tau, j) + W_i(\tau, j)$

For example, referring back to Figure 3.3, and before a request for β_1 appears at any node, the values of $\mathcal{E}_i(\tau, j)$ are calculated as described in Algorithm 3. Take node v_2 for example, then $\mathcal{E}_2(\tau, 1) = W_2(\tau, 1) + W_5(\tau, 1) + W_6(\tau, 1) = 4$. The final expectation values for the rest of the nodes are shown in Figure 3.3.

3.3.4 Effective Caching Duration

The effective caching duration of a content depends on its arrival time. For example, most people read the newspaper in a period of two hours, so the caching duration should be two hours beginning at the arrival of the first request. However, if a new request for the newspaper arrives at a node in the middle of the range and was cached by the algorithm, then the caching duration should be one hour. This requires the broadcast of the first arrival time to all other nodes in the network. The additional overhead incurred by such broadcasting is negligible compared to the reduction of the *Interest* packet broadcasting we achieve through the adoption of en-route caching.

3.3.5 Imperfect Knowledge of the Input Parameters for CRC

Our CRC algorithm achieves asymptotic optimality under the assumption of having exact knowledge of the values of the content popularity expectation, $\mathcal{E}_i(\tau, j)$, and the effective caching duration time, $T_j(\tau)$. Nevertheless, we show the resiliency of our algorithm with respect to errors in the values of $\mathcal{E}_i(\tau, j)$ and $T_j(\tau)$ through simulations.

3.4 Performance Analysis

In this section, we show that any online algorithm has a competitive ratio that is lower bounded by $\Omega(\log(n))$, then we show that our algorithm does not violate the capacity constraints, and achieves a competitive ratio that is upper bounded by $\mathcal{O}(\log(n))$ under realistic settings.

Proposition 1. *Any online algorithm has a competitive ratio which is lower bounded by* $\Omega(\log(n))$.

Proof. We show this proposition by giving an example network, such that the best online algorithm competitive ratio is lower bounded by $\Omega(\log(n))$. Consider a network which consists of $n + 2$ nodes, as shown in Figure 3.6. All contents are originally placed at node S, and node C is the only node with caching capability with a unit cache capacity. All other nodes can request the contents. We consider a 2 time slots system where all contents are to be requested at the beginning of each time slot, though sequentially. Sequentially means that the algorithm has to make a caching decision for a content before considering the next one.

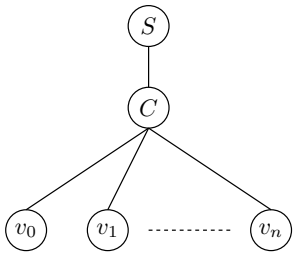

Figure 3.6 Network for lower bound proof.

Consider a $\log(n) + 1$ phases of contents. For each phase $0 \leq i \leq \log(n)$, we have $1/\alpha$ identical contents, each with size $\alpha \ll 1$ and a caching time equal to 2 time slots. Contents in the same phase are destined for the same 2^i nodes. The reason behind considering a 2 time slots system is that when a node caches a content, the traffic saving is considered for future requests.

Let x_i be the fraction of contents stored from phase i and G_i be the traffic saving of the online algorithm gained from phase i, then

$$G_i = x_i 2^i$$

Consider the first k phases, then the online traffic saving of these k phases, denoted by $G(k)$, is

$$G(k) = \sum G_i = \sum_{i=0}^{k} x_i 2^i$$

The offline algorithm will cache the contents from phase k only, gaining a traffic saving of 2^k

Now consider the ratio of the online traffic saving to the offline traffic saving:

$$\begin{aligned}\sum_{k=0}^{\log n} \frac{G(k)}{2^k} &= \sum_{k=0}^{\log n} \sum_{i=0}^{k} \frac{x_i 2^i}{2^k} = \sum_{i=0}^{\log n} \sum_{k=i}^{\log n} x_i 2^{i-k} \\ &= \sum_{i=0}^{\log n} x_i \sum_{k=i}^{\log n} 2^{i-k} \leq 1 * 2 \leq 2\end{aligned}$$

Hence, there exist some k such that $\frac{G(k)}{2^k} \leq \frac{2}{\log n}$. This means that the saving of the offline algorithm is at least within a $\log n$ factor of the savings achieved by any online algorithm. □

Before we start the proof of satisfying the capacity constraints and the upper bound, we need to state the following two assumptions:

$$1 \leq \frac{1}{n} . \frac{\mathcal{E}_i(\tau, j) b_i(j)}{r_j T_j(\tau)} \leq F \qquad \forall j, \forall i \neq S_j, \forall \tau, \tag{3.2}$$

and

$$r_j \leq \frac{\min D_i}{\log(\mu)} \qquad \forall j, \tag{3.3}$$

where F is any constant large enough to satisfy the assumption in (3.2), $\mu = 2(nTF + 1)$, n is the number of caching nodes, and $T = \max(T_j), \forall j$. The assumption in (3.2) states that the amount of traffic savings for a content scales with the content's size and

caching duration. The assumption in (3.3) requires that the caching capacity of any node should be greater than the size of any content, which is a practical condition to assume.

We start by proving that the CRC algorithm does not violate the capacity constraints. After that, we show that CRC achieves a $\mathcal{O}(\log(n))$ competitive ratio. In all of the subsequent proofs, $\tau \in \{t_0(i,j), \ldots, t_0(i,j) + T_j(t_0(i,j))\}$, where $t_0(i,j)$ is the arrival time of β_j at node i.

Proposition 2. *The CRC algorithm does not violate the capacity constraints.*

Proof. Let β_j be the first content that caused the relative load at node i to exceed 1. By the definition of the relative load, we have

$$\lambda_i(\tau, j) > 1 - \frac{r_j}{D_i}$$

using the assumption in (3.3) and the definition of the cost function, we get

$$\begin{aligned}\frac{C_i(\tau,j)}{D_i} &= \mu^{\lambda_i(\tau,j)} - 1 \geq \mu^{1-\frac{r_j}{D_i}} - 1 \\ &\geq \mu^{1-\frac{1}{\log \mu}} - 1 \geq \frac{\mu}{2} - 1 \geq nTF\end{aligned}$$

Multiplying both sides by r_j and using the assumption in (3.2), we get

$$\tfrac{r_j}{D_i} C_i(\tau,j) \geq nTFr_j \geq \mathcal{E}_i(\tau,j) b_i(j) \geq \mathcal{E}_i(\tau,j) d_i(t_0,j)$$

From the definition of our algorithm, β_j should not be cached at node i. Therefore, the CRC algorithm does not violate the capacity constraints. □

The next lemma shows that the traffic saving gained by our algorithm is lower bounded by the sum of the caching costs.

Lemma 1. *Let A be the set of indices of contents cached by the CRC algorithm, and k be the last index, then*

$$2\log(\mu)\sum_{i,j\in A,\tau}[\mathcal{E}_i(\tau,j)d_i(t_0,j)] \geq \sum_{i,\tau} C_i(\tau,k+1) \tag{3.4}$$

Proof. By induction on k. When $k=0$, the cache is empty and the right hand side of the inequality is 0. When β_j is not cached by the online algorithm, neither side of the inequality is changed. Then it is enough to show, for a cached content β_j, that:

$$2\log(\mu)\sum_{i,\tau}[\mathcal{E}_i(\tau,j)d_i(t_0,j)] \geq \sum_{i,\tau}[C_i(\tau,j+1)-C_i(\tau,j)]$$

since summing both sides over all $j \in A$ will yield (3.4).

Consider a node i, the additional cost incurred by caching β_j is given by:

$$\begin{aligned} C_i(\tau,j+1)-C_i(\tau,j) &= D_i[\mu^{\lambda_i(\tau,j+1)}-\mu^{\lambda_i(\tau,j)}] \\ &= D_i\mu^{\lambda_i(\tau,j)}[\mu^{\frac{r_j}{D_i}}-1] \\ &= D_i\mu^{\lambda_i(\tau,j)}[2^{\log\mu\frac{r_j}{D_i}}-1] \end{aligned}$$

Since $2^x - 1 \leq x$ for $0 \leq x \leq 1$ and using the assumption in (3.3)

$$
\begin{aligned}
C_i(\tau, j+1) - C_i(\tau, j) &\leq D_i \mu^{\lambda_i(\tau,j)} [\frac{r_j}{D_i} \log \mu] \\
&\leq r_j \log \mu [\frac{C_i(\tau, j)}{D_i} + 1] \\
&\leq \log \mu [\frac{r_j}{D_i} C_i(\tau, j) + r_j]
\end{aligned}
$$

Summing over τ, i, and the fact that β_j is cached, we get

$$
\begin{aligned}
&\sum_i \sum_\tau [C_i(\tau, j+1) - C_i(\tau, j)] \\
&\quad \leq \log \mu \sum_i \sum_\tau [\frac{r_j}{D_i} C_i(\tau, j) + r_j] \\
&\quad \leq \log \mu [\sum_i \mathcal{E}_i(\tau, j) d_i(t_0, j) + \sum_i \sum_\tau r_j] \\
&\quad \leq 2 \log \mu \sum_i \mathcal{E}_i(\tau, j) d_i(t_0, j)
\end{aligned}
$$

□

In the next lemma, $d_i(\tau, j)$ is defined for the online algorithm.

Lemma 2. *Let Q be the set of indices of contents cached by the offline algorithm, but not the CRC algorithm. Let $l = \arg\max_{j \in Q}(C_i(\tau, j))$. Then*

$$
\sum_i \sum_{j \in Q} \sum_\tau [\mathcal{E}_i(\tau, j) d_i(t_0, j)] \leq \sum_i \sum_\tau C_i(\tau, l)
$$

Proof. Since β_j was not cached by the online algorithm, we have:

$$\sum_{\tau} \mathcal{E}_i(\tau, j) d_i(t_0, j) \leq \sum_{\tau} \frac{r_j}{D_i} C_i(\tau, j)$$

$$\leq \sum_{\tau} \frac{r_j}{D_i} C_i(\tau, l)$$

$$\sum_{i} \sum_{\tau} \mathcal{E}_i(\tau, j) d_i(t_0, j) \leq \sum_{i} \sum_{\tau} \frac{r_j}{D_i} C_i(\tau, l)$$

Summing over all $j \in Q$

$$\sum_{i} \sum_{j \in Q} \sum_{\tau} \mathcal{E}_i(\tau, j) d_i(t_0, j) \leq \sum_{i} \sum_{\tau} C_i(\tau, l) \sum_{j \in Q} \frac{r_j}{D_i} \leq \sum_{i} \sum_{\tau} C_i(\tau, l)$$

Since any offline algorithm cannot exceed a unit relative load, $\sum_{j \in Q} \frac{r_j}{D_i} \leq 1$. □

Combining Lemma 1 and Lemma 2, we have the following lemma.

Lemma 3. *Let A^* be the set of indices of the contents cached by the offline algorithm, and let k be the last index. Then:*

$$\sum_{i,j \in A^*, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j) \leq 2 \log(2\mu) \sum_{i,j \in A, \tau} \mathcal{E}_i(\tau, j) d_i(t_0, j)$$

Proof. The traffic savings of the offline algorithm is given by:

$$
\begin{aligned}
&\sum_{i,j\in A^*,\tau} \mathcal{E}_i(\tau,j)d_i(t_0,j) \\
&\quad= \sum_{i,j\in Q,\tau} \mathcal{E}_i(\tau,j)d_i(t_0,j) + \sum_{i,j\in A^*/Q,\tau} \mathcal{E}_i(\tau,j)d_i(t_0,j) \\
&\quad\leq \sum_{i,j\in Q,\tau} \mathcal{E}_i(\tau,j)d_i(t_0,j) + \sum_{i,j\in A,\tau} \mathcal{E}_i(\tau,j)d_i(t_0,j) \\
&\quad\leq \sum_{i,\tau} C_i(\tau,l) + \sum_{i,j\in A,\tau} \mathcal{E}_i(\tau,j)d_i(t_0,j) \\
&\quad\leq \sum_{i,\tau} C_i(\tau,k+1) + \sum_{i,j\in A,\tau} \mathcal{E}_i(\tau,j)d_i(t_0,j) \\
&\quad\leq (2\log\mu + 1) \sum_{i,j\in A,\tau} \mathcal{E}_i(\tau,j)d_i(t_0,j) \\
&\quad\leq 2\log(2\mu) \sum_{i,j\in A,\tau} \mathcal{E}_i(\tau,j)d_i(t_0,j)
\end{aligned}
$$

□

Note that $d_i(\tau,j)$ in the previous lemmas is defined by the online algorithm. In order to achieve optimality using this proof technique, $d_i(\tau,j)$ of the online algorithm should be equal to $d_i(\tau,j)$ of the offline algorithm. In the next two corollaries, we show cases where $d_i(\tau,j)$ of the online algorithm is equal to $d_i(\tau,j)$ of the offline algorithm.

Corollary 1. *When there is only one caching node in every path, then $d_i(\tau,j)$ of the online algorithm is equal to $d_i(\tau,j)$ of the offline algorithm, and our algorithm achieves asymptotic optimality.*

Corollary 2. *When every node in the path shares the same caching decision, then $d_i(\tau,j)$ of the online algorithm is equal to $d_i(\tau,j)$ of the offline algorithm, and our algorithm achieves asymptotic optimality.*

3.5 Extensions to CRC Algorithm

In this section, we provide some extensions to the CRC algorithm. We show the efficiency of these extensions with respect to currently deployed caching schemes through extensive simulations.

3.5.1 Energy-CRC

The basic CRC algorithm measures the traffic savings for caching a content based on the number of hops. In this section, we provide an extension for the basic CRC algorithm where the savings are measured based on the energy saved on the upstream path from the caching node up to the first node that already has the content in its cache.

The settings for the Energy-CRC algorithm are the same for the basic CRC algorithm except for the definitions of $b_i(j)$ and $d_i(\tau, j)$, where we define $b_i(j)$ as the energy consumption from S_j to node i, and $d_i(\tau, j)$ as the energy consumption on the path from the first node that is currently caching β_j to node i along the path to S_j at time τ.

Specifically, let $\alpha_{(u,v)}$ denote the energy consumption to transfer a unit-size content from node u to node v via a direct link, then the energy consumption to transfer β_j at time τ from node w to node i, where w is the first node along the path from node i to S_j that is currently caching β_j, is given by:

$$d_i(\tau, j) = \sum_{\substack{(u,v): \\ (u,v) \in Path(w,i)}} (r_j \alpha_{(u,v)}).$$

Based on the values of $d_i(\tau, j)$, we apply the basic CRC algorithm.

In the case where renewable energy is used to power the caching nodes, the objective will be to reduce the amount of the consumed non-renewable energy. Therefore, the definition of $d_i(\tau, j)$ changes to reflect the new objective. We measure the traffic savings based on how much non-renewable (brown) energy is saved.

Specifically, let $gr_u(\tau)$ denotes the amount of available renewable energy at node u at time τ, then:

$$d_i(\tau, j) = \sum_{\substack{(u,v) \\ (u,v) \in Path(w,i)}} (\max\{r_j \alpha_{(u,v)} - gr_u(\tau), 0\}).$$

We assume that every caching node has a prior estimation of how much renewable energy will be available in the near future.

3.5.2 Replacement-CRC

The basic CRC algorithm provides quality of service guarantees for content providers by not replacing their contents once they are cached. Content providers, in return, are charged to provide incentives for the caching nodes based on the caching policy discussed in Section 3.3.1. In this section, we present an extension for the basic CRC algorithm that allows content replacement.

The settings for Replacement-CRC are the same as for the basic CRC algorithm. However, there is no restriction on keeping a content β_j in the cache of node i for the whole effective caching duration time $T_j(\tau)$, as β_j may be replaced by another content.

We present the details of the Replacement-CRC algorithm in algorithm 4.

Algorithm 4 states that if the traffic savings gained by caching a new content β_j is greater than the caching cost at node i, then the algorithm decides to cache. Otherwise,

Algorithm 4 Replacement-CRC

A new request for β_j appears at node i at time t_0
$\forall \tau \in \{t_0, \ldots, t_0 + T_j(t_0)\}$, Compute $\lambda_i(\tau, j)$, $C_i(\tau, j)$
if $\sum_\tau \mathcal{E}_i(t_0, j) d_i(t_0, j) \geq \sum_\tau \frac{r_j}{D_i} C_i(\tau, j)$ **then**
    Cache β_j at node i
    $\tau_0(i, j) = t_0(i, j)$
    $\forall \tau \in \{t_0, \ldots, t_0 + T_j(t_0)\}, \lambda_i(\tau, j+1) = \lambda_i(\tau, j) + \frac{r_j}{D_i}$
else
    $\forall \beta_k \in Cache_i(t_0) \cup \beta_j, \forall \tau \in \{t_0, \ldots, t_0 + T_k(t_0)\}$, Compute
    $\lambda_i^k(\tau, j) = \lambda_i(\tau, j) + \frac{r_j}{D_i} - \frac{r_k}{D_i}$
    $C_i^k(\tau, j) = D_i[\mu^{\lambda_i^k(\tau, j)-1}]$
    if $\lambda_i^k(\tau, j) \leq 1$ **then**
        $Diff(k) = \sum_\tau \mathcal{E}_i(\tau_0, k) d_i(\tau_0, k) - \sum_\tau \frac{r_j}{D_i} C_i^k(\tau, j)$
    $l = \text{argmin}_k(Diff)$
    if $l \neq j$ **then**
        Replace β_l with β_j
        $\forall \tau \in \{t_0, \ldots, t_0 + T_j(t_0)\}, \lambda_i(\tau, j+1) = \lambda_i^l(\tau, j)$

we compare the difference between the traffic savings and the caching costs for every $\beta_k \in Cache_i(\tau)$, if it is replaced by β_j without violating the capacity constraints. We then choose the content with the minimum difference to replace with β_j.

3.5.3 Energy-CRC with Replacement

This extension combines Energy-CRC with Replacement-CRC, where the traffic savings are measured based on the brown energy savings, and where content replacement is allowed. We show the efficiency of this extension against currently deployed caching schemes such as Least Recently Used (LRU) through extensive simulations.

3.6 Simulation Results

In this section, we compare our CRC algorithm to some of the existing caching schemes.

3.6.1 Settings

We simulate the following caching schemes:

1. CRC: This scheme represents our basic algorithm.
2. CRC Version 2: This is similar to the CRC scheme, Version 1, except that we retrieve the content from the closest node that has the content in its cache, not necessarily along the path to the content's source.
3. Cache All: This scheme caches every new content arriving at a caching node, as long as there is enough residual capacity to cache the new content.
4. Random Caching Version 1: In this scheme, when a request for a content arrives at node i, the caching probability of the content depends on the content's popularity at node i. The popularity of a content β_j at node i denoted by Pop_j, is defined as the ratio of the number of requests for β_j coming from the subnetwork connected to node i denoted by N_i^j, to the total number of non-caching nodes in the subnetwork connected to node i denoted by N_i. Mathematically speaking, $Pop_j = N_i^j/N_i$. If we choose a uniform random number x between [0,1], and $x \leq Pop_j$, then the content β_j is cached if there is enough room for it in the cache. Otherwise, the content is not cached.
5. Random Caching Version 2: This is similar to Random Caching Version 1, except that the caching probability of the content depends on the content's popularity at node i, scaled by the fraction of the available residual capacity to the total capacity in the cache of node i denoted by f_i, *i.e.*, if we choose a uniform random number x between [0,1], and $x \leq f_i \times Pop_j$, then the content β_j is cached if there is enough room for it in the cache. Otherwise, the content is not cached.

For every caching node i in the network, we assign a cache capacity D_i that is uniformly chosen in the range of [750, 1000] GB. The number of the non-caching nodes connected to the caching node i is chosen uniformly at random in the range of 10 to 90 nodes.

For every content, we randomly chose one of the nodes to act as the source. Each content has a size chosen randomly in the range of [100, 150] MB. The starting effective time of the content is chosen randomly. The end time is also chosen randomly within a fixed interval from the starting time. If the end time exceeds the end time of the

simulation, it is adjusted to be equal to the end time of the simulation. The simulation interval is chosen to be 1000 time slots.

3.6.2 Results on Random Topologies

We start our evaluation on random backbone topologies, in which the caching nodes are generated as a random topology.

We simulate the effect of the number of caching nodes n in the network for three cases, $n = 30$, $n = 50$, and $n = 100$ nodes. For each case we use 10 random topologies, and report the average performance. We fix the effective caching duration to 150 slots and the number of contents to 10000 contents to solely show the effect of increasing the number of nodes on the performance of the CRC algorithm. The results are shown in Figure 3.7(a).

As can be seen from this figure, increasing the number of the caching nodes will result in better performance in all schemes since more contents can be cached. Another observation from the figure is that the performance of CRC schemes increases at a higher rate than other schemes as we increase the number of the nodes in the network. This shows that our scheme greatly benefits from adding more caching nodes to the network. It is also aligned with the property of asymptotic optimality of our scheme. Conversely, not much improvement can be seen from the other schemes when the number of nodes is increased in the network.

We simulate the effect of changing the number of contents from 2000 to 10000. The results are averaged over 10 runs and are shown in Figure 3.7(b). The reason that the performance of the Cache All, Random 1, and Random 2 schemes increases and

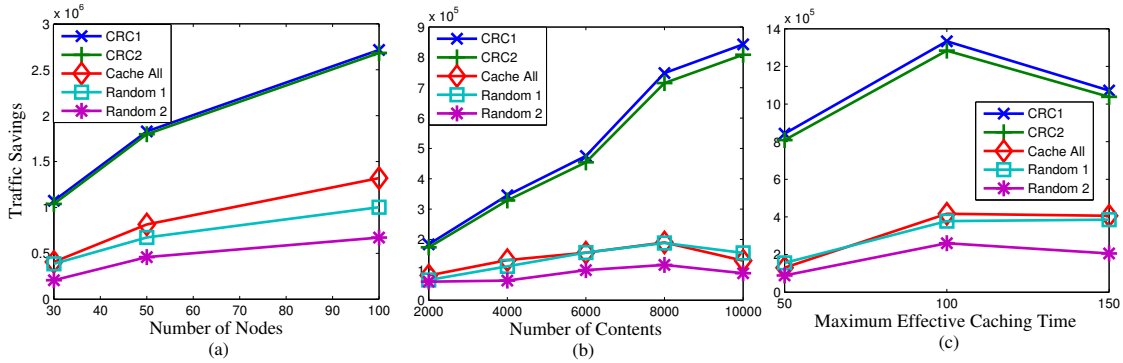

Figure 3.7 Effects of different factors on the performance of the random topologies.

then decreases is that there is a saturation point after which the caches of the network cannot handle the requests. Contrariwise, Our scheme reserves the cache capacity for contents with higher traffic savings, and achieves an improvement of 2 to 3-fold in terms of traffic savings.

Figure 3.7(c) shows the effect of the maximum effective caching duration for three cases, 50, 100, and 150 time slots. In this scenario, the difference between the start and end times for each content is drawn randomly from $\{1, \ldots, \max .caching\ duration\}$. The reason that the traffic savings decrease as the maximum effective caching duration increases after a certain point is that contents are cached for a longer period, so future contents are less likely to find enough residual capacity at the caching node.

In all of the results in Figure 3.7, the performance of CRC Version 2 is always less than the performance of CRC Version 1. This is because CRC Version 2 deviates from the settings under which we achieve optimality.

So far our performance measure was the traffic saving. In Figure 3.8, we measure the cost in terms of total number of hops to satisfy all of the requests. The results in Figure 3.8 are for a random topology with 100 caching nodes, the number of contents

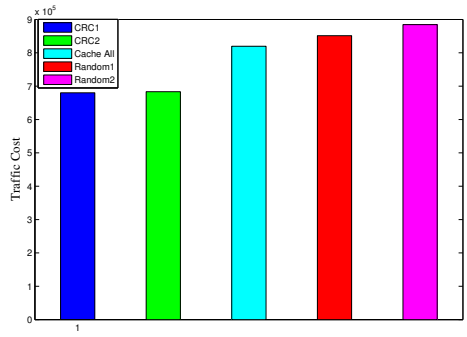

Figure 3.8 Traffic cost.

is 10000, and the maximum effective caching duration is 150 slots. The results in the figure show that even when we measure the performance in terms of the total cost, our scheme reduces the cost by the range of 30% to 50%.

In Figure 3.9 we measure the per topology improvement for all schemes with respect to Random Caching Version 2 scheme. Here, we measure the performance of all schemes for 100 different random topologies. For each topology, we normalize the performance of all schemes with respect to the performance of Random Caching Version 2. Denote the performance of the CRC scheme and Random Caching Version 2 scheme for topology s as $P_{CRC}(top_s)$ and $P_{Random2}(top_s)$, respectively. We compute the normalized performance of CRC scheme with respect to Random Caching Version 2 scheme for topology s as $R_{CRC}(top_s) = P_{CRC}(top_s)/P_{Random2}(top_s)$. After that, the empirical CDF of the vector $R_{CRC} = [R_{CRC}(top_1), R_{CRC}(top_2), \ldots, R_{CRC}(top_{100})]$ for the 100 random topologies is plotted. We do the same process for the other two schemes. The results in this figure show that our scheme experiences about 4 times the improvements as that by Random Caching Version 2.

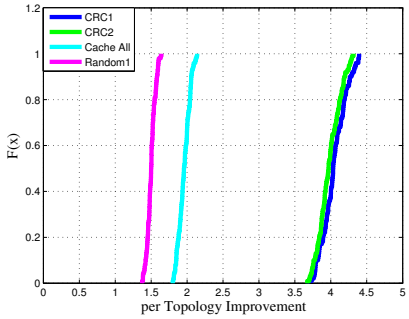

Figure 3.9 Empirical CDF of the per topology improvement for random topologies with respect to random caching version 2.

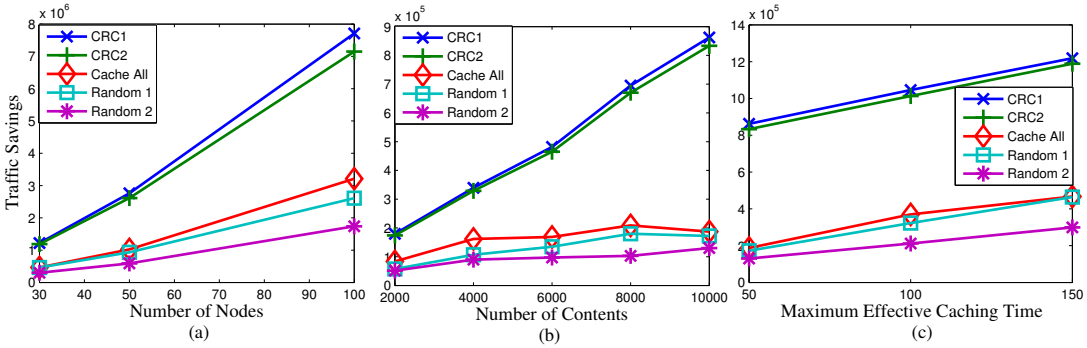

Figure 3.10 Effects of different factors on the performance of the small-world topologies.

3.6.3 Results on a Small-world Generated Topology

In [50] it is shown that the Internet topology exhibits a small-world structure defined in [51]. In this section, we perform simulations based on the small world-structure.

Figure 3.10 is similar to Figure 3.7, but for the small-world topologies. The results follow the same trend as the results for the random topologies except for two differences. The first difference is that is that CRC Version 1 achieves better performance than CRC Version 2 as we increase the number of nodes. The second difference is that all of the schemes performances increase with increasing the effective caching time. One of the reason is due to the sparsity of the small-world topologies, which results in the fact that the requests are distributed over multiple domains inside the topology.

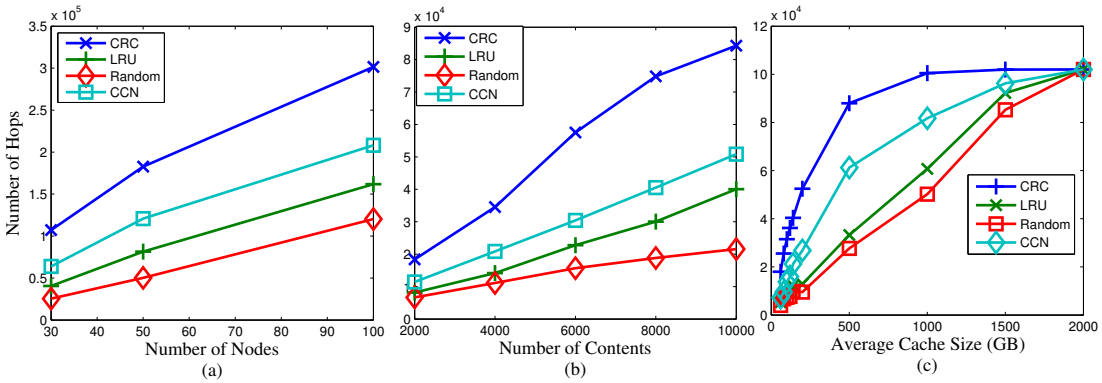

Figure 3.11 Effects of different factors on the performance of different replacement schemes.

3.6.4 Results for Replacement-CRC

We compare the performance of Replacement-CRC against the following schemes:

1. Least Recently Used (LRU): In this scheme, when a request for a content β_j appears at node i, Least Recently Used replacement is performed at all nodes along the path from node i to the source of the content β_j.

2. Random Replacement: In this scheme, when a request for a content β_j appears at node i, every node along the path from node i to the source of the content β_j will randomly choose a cached content to be replaced with β_j, as long as the capacity constraints are satisfied.

3. CCN: This scheme represents the Content Centric Network as described in [16], where a request for a content is broadcasted until the closest node with a copy of the content in its cache is found. The content then follows the path from the closest node to the requester, and all nodes along that path caches the content as long as the capacity constraints are satisfied, or performs replacement using LRU if content replacement is needed.

We use the same settings as described in Section 3.6.1, and we simulate the effect of increasing the number of caching nodes in the network, the effect of increasing the number of contents, and the effect of increasing the cache capacity of the caching nodes. The results are shown in Figure 3.11.

Figure 3.11(a) shows the performance of all schemes as we increase the number of the caching nodes in the network. From this figure, the performance of all schemes

increases with increasing the number of caching nodes. This is because adding more caching nodes will increase the overall caching capacity of the network, which results in more cached contents. Moreover, as the topology grows with adding more nodes, the average distance between the nodes in the network increases. This figure also shows that Replacement-CRC outperforms the existing replacement schemes by 30% to 60%.

Figure 3.11(b) shows the performance of all schemes as we increase the number of contents. As we increase the number of contents, the performance of all schemes increases since more contents are available for caching. Replacement-CRC achieves better performance than the other schemes, since it is able to identify the contents with higher traffic savings and the replacement is done less frequently than the other schemes.

In Figure 3.11(c), we investigate the effect of increasing the caching size of the caching nodes on the performance of all schemes. We increased the caching size of each node until we reach a saturation point, where all of the nodes are able to cache all of the contents without the need for replacement. At this saturation point, all schemes achieves the same traffic savings. Another observation from this figure is that the performance of Replacement-CRC at 500GB is similar to the performance of the other schemes at 1500GB. This means that Replacement-CRC can achieve the same performance of the other schemes with only 30% of the cache capacity.

3.6.5 Energy-CRC with Replacement vs. Cache Size

We investigate the effect of varying the cache size on the performance of Energy-CRC with replacement algorithm as well as currently deployed caching schemes. The

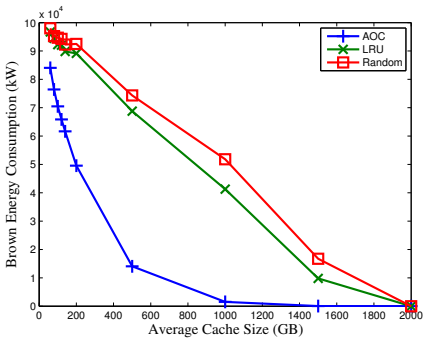

Figure 3.12 Performance of CRC algorithm vs. cache size.

simulation was run on a world-wide topology consisting of 41 nodes spanning 41 different cities around the world as reported in [52]. The distances between the nodes were taken from [53] and the energy consumption for transferring data was taken from [54]. The amount of renewable energy available at each node were taken from [55]. Figure 3.12 shows the brown energy consumption for the Energy-CRC with replacement, Least Recently Used (LRU), and Random replacement vs. the average cache size. As we increase the average cache size of the caching nodes, the caching nodes can cache more contents and achieve lower brown energy consumption, until we reach a point where every caching nodes can cache all the contents and reach the lowest brown energy consumption. From this figure, we see that our algorithm achieves a maximum gain of 60% over other schemes when the average cache size is 500 GB. This is because our algorithm reserves the cache space for contents with high brown energy consumption. This means that our algorithm can lower the brown energy consumption without the necessity of excessive increment in the cache size.

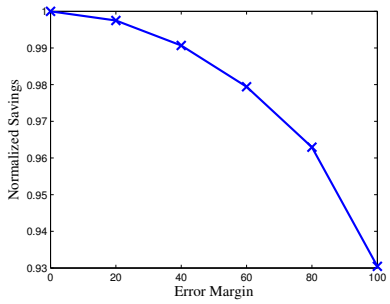

Figure 3.13 Performance of CRC algorithm vs. errors in expectation values.

3.6.6 CRC vs. Error Margins in Input Parameters

We investigate the effect of having error margins in estimating the expectation values and in estimating the maximum effective caching time on the performance of the basic CRC algorithm. Although data mining techniques or stochastic process modeling based on the history [37] can be used to provide good expectation values, we run the simulation as the error margin is changed from 0% to 100%. Denote the error margin by ϵ, then the new values of the expectation $\hat{\mathcal{E}}_i(\tau, j)$ is chosen randomly from a uniform distribution in the range $[(1-\epsilon) \times \mathcal{E}_i(\tau, j), (1+\epsilon) \times \mathcal{E}_i(\tau, j)]$, and the new values of the effective caching time $\hat{T}_j(\tau)$ is chosen randomly from a uniform distribution in the range $[(1-\epsilon) \times T_j(\tau), (1+\epsilon) \times T_j(\tau)]$. Since the basic CRC algorithm achieves asymptotic optimality when having perfect knowledge of the expectation and effective caching time values, introducing errors in these values will cause the performance of the CRC algorithm to decrease.

Figure 3.13 shows the performance of the CRC algorithm versus error margins in estimating the expectation values, while Figure 3.14 shows the performance of the CRC algorithm versus error margins in estimating the effective caching time. From these figures, we see that our algorithm is more sensitive to error margins in estimating

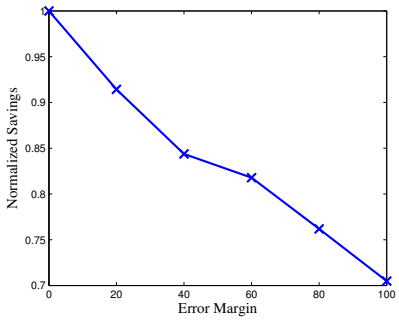

Figure 3.14 Performance of CRC algorithm vs. errors in effective caching time.

the effective caching time. We believe this is due to the adaptability of our algorithm to error margins in estimating the expectation values, that even with such errors, the actual number of requests will still be served. However, error margins in estimating the effective caching time have greater effect on the performance of the basic CRC algorithm, since once a content is cached, the content has to stay in the cache for the whole effective caching duration time.

3.7 Conclusion

Caching at intermediate nodes has the advantage of bringing the contents closer to the users, which results in traffic offloading from the origin servers and lower delays. To achieve this, caching schemes such as en-route caching and CCN have been investigated. Unlike CCN, the use of en-route caching does not require major changes to the TCP/IP model. Previous works have studied en-route caching under offline settings to achieve the optimal content placement strategy. In this work, we study the framework of en-route caching under online settings.

Under this framework, we characterize the fundamental limit for the ratio of the performance of the optimal offline scheme to that of any online scheme. The offline

scheme has a complete knowledge of all of the future requests, while the online scheme does not possess such knowledge. We also design an efficient online scheme and prove that the developed online scheme achieves optimality as the number of nodes in the network becomes large. Moreover, we introduce some extensions to the algorithm. Our simulation results affirm the efficiency of our scheme and its extensions.

CHAPTER 4

ONLINE ALGORITHM FOR CACHING IN MULTI-CELL COORDINATED SYSTEMS

4.1 Introduction

The proliferation of content delivery services in the recent years motivated changes to the operations of cellular networks, as the current infrastructure cannot cope with this increase. One way to handle this challenge is to introduce caching at the base stations. Caching at the base stations can reduce the data traffic going through the backhaul links, reduce the time required for content delivery, and help in smoothing the traffic during peak hours. Thus, providing good caching techniques is of high importance. Note that the price of data storage devices is dramatically decreasing year by year. In this chapter[1], we present an online algorithm for content caching in Multi-cell coordinated networks, where the network consists of multiple base stations that can collaborate with each other to satisfy requests for contents made by the users of the cellular network.

We consider collaborative caching at the base stations from a different perspective. Our objective is to minimize the overall cost paid by the Content Providers (CPs). We assume that caching a content at a base station incurs two types of costs. The first type is the storage cost, where CPs have to pay to the Cellular Network Operator (CNO) in exchange for caching their contents. This is motivated by the increasing trend of using in-network cloudlets, services, and middleboxes, in which the storage and computations are performed at small clouds installed in the routers or the base

[1]The work of this chapter has been published in [56]

stations of the network [57–60]. Moreover, the caching costs paid by the CPs motivate the CNOs to perform caching by providing them with an extra source of income.

The second type of cost is what we call User Attrition (UA) cost [61]. This cost represents the expected cost of losing users that are switching to other CPs because the users are not getting their desired Quality of Service (QoS). This is caused by the fact that the requested content is cached far away from the users. For example, users who are experiencing high delays when streaming a video from one CP may switch to another CP, which causes losses for the former CP. These two types of costs yield a tradeoff on where the CPs choose to cache their contents in order to minimize the total cost. In this chapter, we formulate the problem of caching in a multi-cell coordinated system as an optimization problem that minimizes the overall cost paid by the CPs, while satisfying the users' demands.

In the formulated optimization problem, we assume the exact knowledge of the contents popularities. Based on this knowledge, a proactive offline algorithm for collaborative caching can achieve the optimal solution, similar to [10, 11, 44, 45]. Since in real life scenarios this knowledge is unavailable, an online algorithm is needed for caching at the base stations. In the online algorithm, a decision for caching at a base station is made when a content is requested, and the caching decision cannot be changed in the future because the CP has already paid the caching cost. To measure the performance of the online algorithm, we use the concept of competitive ratio. The competitive ratio is the ratio of the performance of the online algorithm to the performance of the optimal offline algorithm. In this chapter, we present an online algorithm with a competitive ratio of $\mathcal{O}(\log(n))$, where n is the total number of requests

in the cellular network. The competitive ratio we obtain is close to the lower bound of $\Omega(\frac{\log(n)}{\log\log(n)})$ which we prove in Section 4.4.6, since $\log\log(n)$ is small even when n is large (*i.e.*, $\log\log(10^6) \approx 4.3$).

4.2 Settings

We consider a cellular network consisting of a set $\mathcal{K} = \{1, 2, \ldots, k, \ldots, K\}$ of cache-capable base stations connected to each other via backhaul links. The backhaul links also serve as a connection to the Internet through the cellular system gateway. In the rest of the chapter, we use the words base station and cache interchangeably. Let $K+1$ denote the index of the contents providers servers located in the Internet. Figure 4.1 provides an example of our system model. We have $\mathcal{M} = \{1, 2, \ldots, j, \ldots, M\}$ contents with sizes $\mathcal{S} = \{s_1, s_2, \ldots, s_j, \ldots, s_M\}$ that can be requested by the users connected to the base stations. Let γ_{ij} denote the number of requests for the j-th content generated by users in the i-th base station.

Due to the dramatic decrease in data storage prices, we assume that the cache capacity of each base station is unlimited. However, there is a unit cost f_{kj} associated with caching the j-th content at the k-th base station. Having a caching cost will limit the number of contents cached at a base station. Moreover, as explained in Section 4.1, there is an increasing trend of using in-network cloudlets in which the base station itself becomes a small cloud. Also, due to the fast development and cost reduction of storage devices, the cache size can be very large with low cost.

Let T_{ij}^k denote the UA cost associated when the i-th base station retrieves the j-th content from the k-th base station, and let T_{ij}^{K+1} denote the UA cost associated when

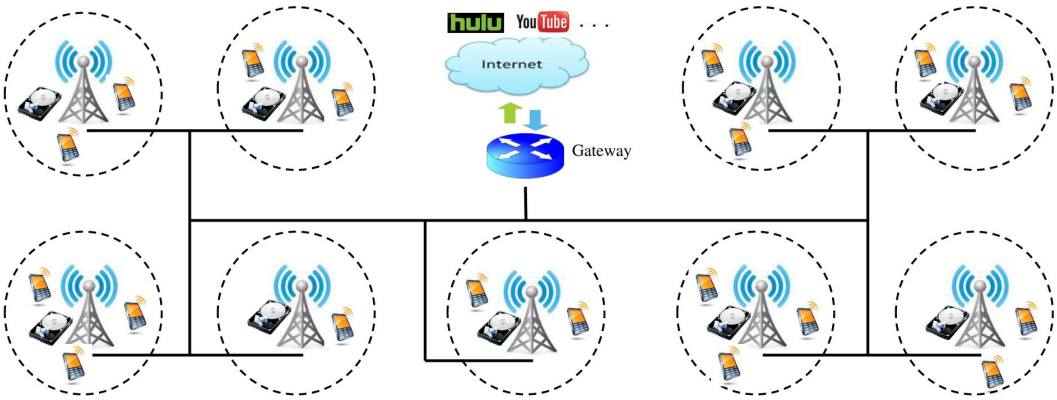

Figure 4.1 Collaborative multi-cell coordinated system.

the i-th base station retrieves the j-th content from the Internet. If we associate a cost between any two directly connected base stations, then for any two base stations i, k, T_{ij}^{k} can be computed using the minimum cost path between i and k, and thus satisfies the triangle inequality (*i.e.*, $T_{ij}^{k} \leq T_{ij}^{k'} + T_{k'j}^{k}$).

Our objective is to find a caching setup that minimizes the aggregated caching and UA costs while satisfying the users' demands. We introduce the formulation of the optimization problem in the next section.

4.3 Problem Formulation

In this section, we formulate the problem of collaborative multi-cell coordinated system, followed by the proof of the problem's NP-completeness.

4.3.1 The Formulation

Before presenting our formulation, we introduce the following variables:

$$Y_{kj} = \begin{cases} 1 & \text{if the } j\text{-th content is cached} \\ & \text{at the } k\text{-th base station.} \\ 0 & \text{otherwise.} \end{cases}$$

$$X_{ij}^{k} = \begin{cases} 1 & \text{if the } i\text{-th base station retrieves the} \\ & j\text{-th content from the } k\text{-th base station.} \\ 0 & \text{otherwise.} \end{cases}$$

$$X_{ij}^{K+1} = \begin{cases} 1 & \text{if the } i\text{-th base station retrieves the} \\ & j\text{-th content from the Internet.} \\ 0 & \text{otherwise.} \end{cases}$$

We formulate the problem as the following Integer Linear Program (ILP):

$$\min \sum_{i=1}^{K} \sum_{k=1}^{K+1} \sum_{j=1}^{M} T_{ij}^{k} X_{ij}^{k} \gamma_{ij} s_j + \sum_{i=1}^{K} \sum_{j=1}^{M} f_{kj} Y_{kj} s_j$$

Subject to

$$X_{ij}^{k} \leq Y_{kj}, \quad \forall i, j, k \tag{4.1}$$

$$\sum_{k=1}^{K+1} X_{ij}^{k} \geq 1_{\{\gamma_{ij}>0\}} \quad \forall i, j \tag{4.2}$$

In the objective function of the above problem, the first term is the total UA cost, and the second term is the total caching cost. The first set of constraints ensures that a content can be retrieved from a base station only if the content is in the cache of that base station. The second set of constraints ensures that if a content is requested,

then the content is served either from the cache of the local base station, the cache of a neighboring base station, or from the Internet.

In the following section, we present the proof of the problem's NP-completeness.

4.3.2 NP-Completeness Proof

In this section, we show that the ILP optimization problem presented in the previous section is NP-complete by proving the following theorem:

Theorem 1. *The ILP optimization problem is NP-complete.*

Proof. Since we have $M(K^2 + K)$ constraints, we can easily check the feasibility of any given solution in polynomial time by checking that the set of constraints (4.1)-(4.2) are not violated, thus the problem is in NP.

To prove that the problem is NP-hard, we reduce the set cover problem, which is known to be an NP-complete problem, to an instance of our problem. The set cover problem is defined as follows: Given a set of elements $U = \{u_1, u_2, \ldots, u_N\}$ called the universe, and a family of subsets of the elements in the universe $B = \{b_1, b_2, \ldots, b_L\}$, where each subset b_k has a cost a_k. The objective is to find a collection of subsets of B, whose union is the universe, and its total cost is minimized.

The reduction from the set cover problem to our problem is done as follows: (1) The number of contents in our problem is set to 1. (2) Each element u_i in the set cover problem is mapped to a base station requesting the content in our problem. (3) The caching cost of the k-th base station f_k is set to a_k. (4) The demand γ_i is set to 1 for all base stations. (5) The UA cost T_i^k is set to 0 if $u_i \in b_k$ and is set to $2a_k$ otherwise.

Note that due to the reduction from the set cover problem to our problem, all the elements in the set cover problem are covered iff the total UA cost of the solution to our problem is 0. Now we prove that there exists a solution to the set cover problem of cost no greater than $\mathcal{A}$ iff there exists a solution to our problem with a cost no greater than $\mathcal{A}$.

The first direction is easy to see. If there exists a solution to the set cover problem with cost $\mathcal{A}$, then the sets form the caches and the total cost of our problem is $\mathcal{A}$. To prove the other direction, suppose we have a solution to our problem with a cost of $\mathcal{A} = \mathcal{A}^{Caching} + UA^{total}$, where $\mathcal{A}^{Caching}$ is the total caching cost and UA^{total} is the total UA cost. Then we have the following two cases:

- The total UA cost is equal to 0. In this case, the selected caches yield the collection of sets for the set cover problem with cost $\mathcal{A}$.
- The total UA cost is greater than 0. This means that there are some elements in the corresponding set cover problem that are still not covered. In this case, for each base station i whose incurring a non-zero UA cost, we cache the content at a new cache k (*i.e.*, select a new subset b_k) such that $T_i^k = 0$. Since $f_k + T_i^k = a_k + 0 \leq 2a_k$, the new total cost $\mathcal{A}' < \mathcal{A}$. Then go back to case 1.

□

4.4 Online Algorithm

Before we present the online algorithm, we point out the following observation. Due to the assumption that the cache capacities of the base stations are unlimited, the decision of caching a content at a base station is independent from the other contents, so we can view our problem as M independent caching subproblems. Even with this decomposition, the proof of NP-completeness presented in Section 4.3.2 still holds for

every subproblem, since we reduce the set cover problem to an instance of our problem that has one content. In the sequel, we only consider the j-th content, and hence, the content-index j is omitted in the subscript of the notations throughout the rest of the chapter. Moreover, in the online algorithm and in the proof of the competitive ratio, every term will be multiplied by the content size s_j, and hence, we set $s_j = 1$.

4.4.1 The Algorithm

The Online Collaborative Caching (OCC) algorithm is presented in Algorithm 5. We note that Algorithm 5 is for a specific content. For multiple contents, each content will have its own instance of the algorithm.

Throughout the algorithm, the following notations are used:

- W: the set of base stations already caching the content.
- V: the set of requests processed so far by the algorithm.
- $p(k)$: the potential function of the k-th base station.
- For a cache u and a request v arriving at the i-th base station, $d(u, v) = T_i^u$
- For a set of caches $\mathcal{X}$ and a request v arriving at the i-th base station, $d(\mathcal{X}, v) \equiv \min_{k \in \mathcal{X}} d(k, v)$.
- $[x]^+ \equiv \max\{x, 0\}$.

The functions *initializePotentials()*, *updatePotentials()*, and *computeNewPotentials()* used in Algorithm 5 are presented in Algorithm 6.

The intuition behind Algorithm 5 is the following: we define a potential function for each base station. When a new request for a content arrives at a base station, the algorithm updates the potential function of each base station, which represents the total

Algorithm 5 Online Collaborative Caching (OCC)

$W \leftarrow \{K+1\}, V \leftarrow \phi, Cost = 0$, initializePotentials()
for each new request v arriving at the i-th base station **do**
  if The i-th base station is already caching the content **then**
    Satisfy the request
  else
    $V \leftarrow V \cup \{v\}$
    updatePotentials(W, v)
    $w \leftarrow \arg\max_k (p(k) - f_k)$
    if $p(w) - f_w > 0$ **then**
      $W \leftarrow W \cup \{w\}$
      $Cost = Cost + f_w$
      computeNewPotentials(W, V)
    assign v to $\alpha = \text{argmin}_{k \in W}\, d(k, v)$
    $Cost = Cost + T^i_\alpha$

Algorithm 6 Functions used in Algorithm 5

initializePotentials()
for all $k \in \mathcal{K} \cup \{K+1\}$ **do**
  $p(k) = 0$

updatePotentials(W, v)
for all $k \in \mathcal{K} \cup \{K+1\}$ **do**
  $p(k) = p(k) + [d(W, v) - d(k, v)]^+$

computeNewPotentials(W, V)
for all $k \in \mathcal{K} \cup \{K+1\}$ **do**
  $p(k) = \sum_{v \in V} [d(W, v) - d(k, v)]^+$

UA cost of the requests seen so far when that base station retrieves the content from the base station with the lowest UA cost (probably itself). The algorithm decides to cache the content at a base station when the potential of that base station exceeds its caching cost.

4.4.2 Implementation and Complexity

The execution of Algorithm 5 is done by the Mobility Management Entity (MME) of the cellular network, which acts as a centralized controller [62]. Different architectures for cellular networks like Software-Defined Cellular Networks [63] take advantage of the centralized controller. The MME has access to the topology of the cellular network as well as the contents cached at each base station. The content providers pay the operators for running the caching algorithm at the MMEs and storing the contents at the base stations.

When a request for content arrives at a base station, the base station responds with the requested content if it already has a copy of the content in its cache. Otherwise, the base station sends a message indicating the requested content to the MME to execute the algorithm. Based on the available information to the MME (*i.e.*, the topology of the cellular network as well as the contents cached at each base station), the MME runs the algorithm and decides whether the content has to be cached at a new base station w or not. The MME then relays to the new base station w (if any) the decision to cache the requested content and relays to the requesting base station the decision of which base station to retrieve the content from.

Next, we analyze the complexity of Algorithm 5. Recall that K denotes the number of base stations. The *initializePotentials()* subroutine is executed once and has a complexity of $\mathcal{O}(K)$. For every new request, executing the *updatePotentials()* subroutine, finding the base station with the maximum difference between the value of the potential function and the caching cost (line 5 in Algorithm 5), and executing the *computeNewPotentials()* subroutine each has a complexity of $\mathcal{O}(K)$. Note that the *computeNewPotentials()* subroutine is executed only when the content is cached at a new cache, and hence is executed at most K times. Therefore, for n total number of requests, the overall complexity of implementing the algorithm is $\mathcal{O}(Kn + K^2)$.

In order to execute the algorithm, the MME needs to maintain two tables. The first table includes the value of the potential function of each base station. For the second table, it suffices to store the number of requests that arrived at each base station, since the value of $[d(W, v) - d(k, v)]^+$ in the *computeNewPotentials()* subroutine is the same for requests arriving at the same base station. Therefore, the storage requirement needed at the MME is of order $\mathcal{O}(K)$. Note that the size of the tables is independent of the content's size. Therefore, as the size of the content becomes larger, which is the case for future content delivery traffic, the size of the tables will not grow.

4.4.3 Preliminaries

To compute the competitive ratio of the algorithm, we compare the algorithm's cost with the cost of the offline optimal solution. In the optimal offline solution, let W^* denote the set of base stations caching the content. Then, the cost of the offline optimal

solution is given by:

$$\mathcal{C}^* = \sum_{w \in W^*} f_w + \sum_{v \in V} d(W^*, v) \tag{4.3}$$

Let the optimal solution W^* consist of l caches $c_1, c_2, \ldots, c_l$. In the optimal offline solution, each request is satisfied by retrieving the content from a cache. Hence, W^* divides the requests into optimal clusters $C_1, C_2, \ldots, C_l$. For example, if the optimal solution decides to cache a content on three caches c_1, c_4, and c_5, then the first cluster C_1 consists of c_1 and all the base stations retrieving the content from c_1, the second cluster C_2 consists of c_4 and all the base stations retrieving the content from c_4, and so on.

4.4.4 Proof Outline

We start by proving that OCC algorithm maintains the invariant $p(k) \leq f_k$ for all $k \in \mathcal{K} \cup \{K+1\}$ (Lemma 4). Based on Lemma 4 and the triangle inequality, we show that after j requests from cluster C_i, there is a cache where the UA cost from the optimal cache c_i is $\frac{1}{j}[f_{c_i} + 2\sum_{v \in C_i} d(W^*, v)]$ (Corollary 3). This is used to show that the UA cost of OCC is within a logarithmic factor of the total optimal cost $\mathcal{C}^*$ (Lemma 5).

For each new request v, we define a credit $\hat{c}(v)$. We show that $\hat{c}(v) = \min\{d(W, v), \min_k\{f_k - p(k) + d(k, v)\}\}$ (Lemma 6). We then show that the algorithm's total caching cost never exceeds the total credit of the requests in V (Lemma 7). Using Corollary 3, we show that the total credit is within a logarithmic factor of the total optimal cost $\mathcal{C}^*$ (Lemma 8).

4.4.5 The Proof

Lemma 4. $p(k) \leq f_k, \forall k \in \mathcal{K} \cup \{K+1\}$.

Proof. We prove this lemma by induction on the number of requests considered by the online algorithm. For the first request, the invariant holds since $p(k) = 0$ for all $k \in \mathcal{K} \cup \{K+1\}$. We inductively assume that the invariant holds just before a new request v arrives and prove that the invariant holds after v is assigned to retrieve the content from a cache.

Let W be the set of caches that have cached the content, and let V be the set of requests considered so far by the algorithm. Let $p(k) = \sum_{v' \in V} [d(W, v') - d(k, v')]^+ \leq f_k$ be the potential of cache k just before the new request v arrives. Let $p'(k)$ be the potential after the subroutine *updatePotentials()* in the algorithm is executed. Finally, let $p''(k)$ be the potential of cache k after the request v is assigned to a cache. We want to prove that $p''(k) \leq f_k$ for all k.

We have two cases:

- If the new request v does not change W (*i.e.*, the new request did not cause the content to be cached at a new cache), then from the algorithm:
$$p'(k) - f_k \leq 0 \qquad \forall k$$
and $p''(k) = p'(k)$. Therefore, $p''(k) \leq f_k$ for all k.
- If the new request v causes the content to be cached at a new cache w, then
$$\begin{aligned} 0 < p'(w) - f_w &= [d(W, v) - d(w, v)]^+ + p(w) - f_w \\ &\leq [d(W, v) - d(w, v)]^+ \end{aligned}$$
where the first inequality holds because the content is cached at w, the next equality holds from the definition of $p'(w)$, the last inequality holds from the induction hypothesis $p(w) \leq f_w$. Therefore, $[d(W, v) - d(w, v)]^+ \geq p'(w) - f_w > 0$. This implies that $d(W, v) > d(w, v)$, which means that v will be assigned to w and
$$d(W \cup \{w\}, v) = d(w, v) \tag{4.4}$$
From here, we have two cases:

– For all caches k where $d(k,v) < d(w,v)$, we have $[d(W \cup \{w\}, v) - d(k,v)]^+ = d(w,v) - d(k,v)$.
Using (4.4), we get that

$$\begin{aligned} d(W,v) - d(w,v) & \\ &\geq p'(w) - f_w \\ &\geq p'(k) - f_k \\ &= [d(W,v) - d(k,v)]^+ + p(k) - f_k \\ &\geq d(W,v) - d(k,v) + p(k) - f_k \end{aligned}$$

where the second inequality follows from the fact that the content is cached at w, and the first equality follows from the definition of $p'(k)$. Therefore, $d(W,v) - d(w,v) \geq d(W,v) - d(k,v) + p(k) - f_k$. Rearranging the terms we get

$$\begin{aligned} 0 \geq & d(w,v) - d(k,v) + p(k) - f_k \\ \geq & d(w,v) - d(k,v) + \sum_{v' \in V} [d(W,v') - d(k,v')]^+ - f_k \\ \geq & d(w,v) - d(k,v) \\ & + \sum_{v' \in V} [d(W \cup \{w\}, v') - d(k,v')]^+ - f_k \\ \geq & p''(k) - f_k \end{aligned}$$

where the last inequality follows from the definition of $p''(k)$.

– For all caches k where $d(k,v) \geq d(w,v)$, we have $[d(W \cup \{w\}, v) - d(k,v)]^+ = 0$. Therefore,

$$\begin{aligned} p''(k) &= \sum_{v' \in V \cup \{v\}} [d(W \cup \{w\}, v') - d(k,v')]^+ \\ &= \sum_{v' \in V} [d(W \cup \{w\}, v') - d(k,v')]^+ \\ &\leq \sum_{v' \in V} [d(W,v') - d(k,v')]^+ \\ &= p(k) \leq f_k \end{aligned}$$

where the first equality holds from the definition of $p''(k)$, the second equality follows since $[d(W \cup \{w\}, v) - d(k,v)]^+ = 0$, the third inequality follows since $d(W \cup \{w\}, v') \leq d(W,v'), \forall v'$, and the last equality follows from the definition of $p(k)$ before the request v appears.

□

Corollary 3. *Let V be the request set, and W the set of caches caching the content after all requests in V have been considered. Then for every optimal cluster C_i with cache c_i,*

$$|V \cap C_i| d(W, c_i) \leq f_{c_i} + 2 \sum_{v \in C_i} d(W^*, v)$$

Proof. for cache c_i we have:

$$\begin{aligned}
p(c_i) &= \sum_{v \in V} [d(W, v) - d(c_i, v)]^+ \\
&\geq \sum_{v \in V \cap C_i} [d(W, v) - d(c_i, v)] \\
&\geq \sum_{v \in V \cap C_i} [d(W, c_i) - d(c_i, v) - d(c_i, v)] \\
&\geq \sum_{v \in V \cap C_i} d(W, c_i) - 2 \sum_{v \in V \cap C_i} d(c_i, v) \\
&\geq \sum_{v \in V \cap C_i} d(W, c_i) - 2 \sum_{v \in C_i} d(c_i, v)
\end{aligned}$$

where the second inequality is obtained by using the triangle inequality (*i.e.*, $d(W, v) \geq d(W, c_i) - d(c_i, v)$).

Using the invariant $f_{c_i} \geq p(c_i)$ from Lemma 4 and rearranging the terms, we get:

$$|V \cap C_i| d(W, c_i) \leq f_{c_i} + 2 \sum_{v \in C_i} d(W^*, v)$$

where $d(W^*, v) = d(c_i, v)$ for all $v \in C_i$ follows from the definition of cluster C_i. □

In the next lemma, we use Corollary 3 to bound the UA cost incurred by OCC algorithm.

Lemma 5. *Let $\sum_{v\in V} d(W,v)$ denote the total UA cost incurred by OCC algorithm. Then*

$$\sum_{v\in V} d(W,v) \leq \log(n+1) \sum_{w\in W^*} f_w + (2\log(n+1)+1) \sum_{v\in V} d(W^*,v)$$

Proof. Let C_i be an optimal cluster with cache c_i. Let $n_i \equiv |C_i|$ be the number of requests in C_i, and let $v_1, v_2, \ldots, v_{n_i}$ be the requests in C_i in the order considered by the algorithm.

For each request v_j, let W_{v_j} be the set of caches caching the content at v_j's assignment time. Then, using triangle inequality we have

$$d(W_{v_j}, v_j) \leq d(W_{v_j}, c_i) + d(c_i, v_j)$$

From Corollary 3, we have

$$d(W_{v_j}, c_i) \leq \frac{1}{j}[f_{c_i} + 2\sum_{v\in C_i} d(W^*, v)]$$

Therefore

$$d(W_{v_j}, v_j) \leq \frac{1}{j}[f_{c_i} + 2\sum_{v\in C_i} d(W^*, v)] + d(c_i, v_j)$$

Summing over all $v_j \in C_i$ we get that $\sum_{j=1}^{n_i} d(W_{v_j}, v_j)$

$$\begin{aligned}
&\leq [f_{c_i} + 2\sum_{v\in C_i} d(W^*, v)] \sum_{j=1}^{n_i} \frac{1}{j} + \sum_{j=1}^{n_i} d(c_i, v_j) \\
&\leq \log(n_i+1) f_{c_i} + (2\log(n_i+1)+1) \sum_{j=1}^{n_i} d(c_i, v_j)
\end{aligned}$$

The lemma follows by summing over all clusters. □

The next 3 lemmas are used to bound the caching cost incurred by OCC algorithm. To do this, we define a credit $\hat{c}(v)$ to each new request v. We show that $\hat{c}(v) = \min\{d(W,v), \min_k\{f_k - p(k) + d(k,v)\}\}$. Then we show that the total caching cost is upper bounded by the total credits of all requests, which in turn is within a logarithmic factor of the optimal offline cost $\mathcal{C}^*$.

Lemma 6. *For each new request v, $\hat{c}(v) = f_w - p(w) + d(w,v)$ if v causes the content to be cached at w, and $\hat{c}(v) = d(W,v)$ otherwise.*

Proof. Let $p(k)$ denote the potential function of the cache k just before a new request v arrives, and let $p'(k) = p(k) + d(W,v) - d(k,v)$ be the potential function after the subroutine *updatePotential()* is executed.

On the one hand, if the new request v causes the content to be cached at w, then from the algorithm,

$$\begin{aligned} 0 < p'(w) - f_w &= p(w) + [d(W,v) - d(w,v)]^+ - f_w \\ &= p(w) + d(W,v) - d(w,v) - f_w \\ &\geq p(k) + [d(W,v) - d(k,v)]^+ - f_k \\ &\geq p(k) + d(W,v) - d(k,v) - f_k \end{aligned}$$

The first inequality holds because the content is cached at w. The first equality follows from the definition of $p'(w)$. The second equality follows by using (4)(see the proof of Lemma 4). The second inequality holds because $p'(w) - f_w \geq p'(k) - f_k, \forall k$ and from the definition of $p'(k)$. Therefore, $p(w) + d(W,v) - d(w,v) - f_w \geq p(k) + d(W,v) - d(k,v) - f_k$. Rearranging the terms we get that $f_w - p(w) + d(w,v) \leq f_k - p(k) + d(k,v)$.

We also have $p(w)+d(W,v)-d(w,v)-f_w > 0$. Therefore, $\hat{c}(v) = f_w - p(w) + d(w,v)$ if v causes the content to be cached at w.

On the other hand, if the new request v does not cause an additional copy of the content to be cached, then from the algorithm,

$$p'(k) - f_k \leq 0 \qquad \forall k$$

$$p(k) + d(W,v) - d(k,v) - f_k \leq 0 \qquad \forall k$$

$$d(W,v) \leq f_k - p(k) + d(k,v) \qquad \forall k$$

Therefore, $\hat{c}(v) = d(W,v)$. □

In the next lemma, we show that the total caching cost incurred by OCC algorithm is upper bounded by the total credit of the requests in V.

Lemma 7. *Let V be the set of requests, and let W be the set of caches caching the content after all requests in V have been considered. Then*

$$\sum_{w \in W} f_w \leq \sum_{v \in V} \hat{c}(v)$$

Proof. We prove this lemma by a potential function argument. We define the potential function $\Phi = \sum_{v \in V} d(W,v)$ and calculate the change $\Delta\Phi$ in the value of the potential function when a new request v is considered. Let $p(k)$ be the potential of each cache k just before the new request v arrives.

If the new request v does not cause the content to be cached at a new cache (*i.e.*, W is not changed), then $\Delta\Phi = d(W,v) = \hat{c}(v)$ by Lemma 6. Otherwise, if v causes the content to be cached at w, then $d(W \cup \{w\}, v) = d(w,v)$ (recall (4.4) in Lemma 4)

for all $v \in V$, and $d(W, v) - d(W \cup \{w\}, v) = [d(W, v) - d(w, v)]^+$. Therefore,

$$\begin{aligned}\Delta\Phi &= d(w, v) - \sum_{v' \in V} [d(W, v') - d(W \cup \{w\}, v')] \\ &= d(w, v) - \sum_{v' \in V} [d(W, v') - d(w, v')]^+ \\ &= d(w, v) - p(w)\end{aligned}$$

From Lemma 6, we have $\hat{c}(v) = f_w - p(w) + d(w, v) = f_w + \Delta\Phi$. Therefore, $\sum_{v \in V} \hat{c}(v) = \Phi + \sum_{w \in W} f_w$. The lemma follows since $\Phi \geq 0$. □

In the next lemma, we use Corollary 3 to upper bound the total credit of the requests in V.

Lemma 8. *Let V be the set of requests. Then*

$$\sum_{v \in V} \hat{c}(v) \leq (\log(n) + 1) \sum_{w \in W^*} f_w + (2\log(n) + 1) \sum_{v \in V} d(W^*, v)$$

Proof. Let C_i be an optimal cluster with cache c_i and $n_i \equiv |C_i|$ be the number of requests in C_i. For each request $v_j \in C_i$, let W_{v_j} be the set of caches caching the content just before v_j arrives. Note that $d(W^*, v_j) = d(c_i, v_j), \forall v_j \in C_i$.

The credit of each request v_j is $\hat{c}(v_j) \leq \min\{d(W_{v_j}, v_j), f_{c_i} + d(W^*, v_j)\}$ (using Lemma 6). For the first request, $\hat{c}(v_j) \leq f_{c_i} + d(W^*, v_j)$. For the remaining requests $v_j, j \geq 2$, we use Corollary 3 to get

$$\begin{aligned}\hat{c}(v_j) &\leq d(W_{v_j}, v_j) \\ &\leq d(W_{v_j}, c_i) + d(c_i, v_j) \\ &\leq \frac{1}{j-1}[f_{c_i} + 2 \sum_{v_j \in C_i} d(W^*, v_j)] + d(c_i, v_j)\end{aligned}$$

Summing over all v_j, we get

$$\begin{aligned}\sum_{j=1}^{n_i} \hat{c}(v_j) &\leq f_{c_i} + [f_{c_i} + 2\sum_{v_j \in C_i} d(W^*, v_j)] \sum_{j=2}^{n_i} \frac{1}{j-1} \\ &\quad + \sum_{j=1}^{n_i} d(W^*, v_j) \\ &\leq (\log(n_i)+1) f_{c_i} + (2\log(n_i)+1) \sum_{j=1}^{n_i} d(W^*, v_j)\end{aligned}$$

The lemma follows by summing over all clusters. □

Now we are ready to prove the algorithm's competitive ratio

Theorem 2. *The competitive ratio of OCC algorithm is no more than* $4\log(n+1)+2$.

Proof. From Lemma 5, we have

$$\sum_{v \in V} d(W, v) \leq \log(n+1) \sum_{w \in W^*} f_w + (2\log(n+1)+1) \sum_{v \in V} d(W^*, v)$$

and from Lemmas 7 and 8, we have

$$\sum_{w \in W} f_w \leq (\log(n)+1) \sum_{w \in W^*} f_w + (2\log(n)+1) \sum_{v \in V} d(W^*, v)$$

Combining the two bounds we get

$$\begin{aligned}\sum_{w \in W} f_w + \sum_{v \in V} d(W, v) &\leq (2\log(n+1)+1) \sum_{w \in W^*} f_w \\ &\quad + (4\log(n+1)+2) \sum_{v \in V} d(W^*, v) \\ &\leq (4\log(n+1)+2)\mathcal{C}^*\end{aligned}$$

□

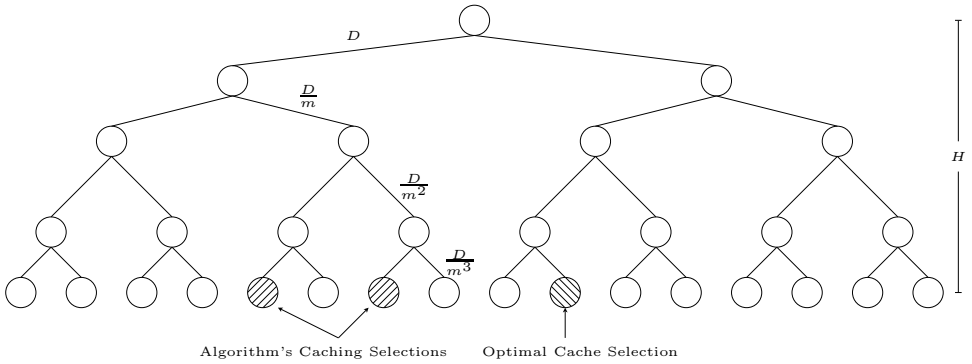

Figure 4.2 Tree structure used in the proof of Theorem 3.

4.4.6 Lower Bound

To prove the lower bound of the competitive ratio of any online algorithm under our settings, we measure the competitive ratio of the online algorithm against an oblivious adversary. For a deterministic algorithm, the adversary knows how the algorithm works, so the adversary can always generate an input sequence such that the deterministic algorithm performs worst on that input. For a randomized algorithm, the adversary knows the algorithm's code, but does not know the randomized result of the randomized algorithm, so the performance of the randomized algorithm is not worse than the performance of the deterministic algorithm against the same adversary. This means that a lower bound on the competitive ratio of the randomized algorithm is also a lower bound on the competitive ratio of the deterministic algorithm. In the next theorem, we show that the competitive ratio of any randomized online algorithm is lower bounded by $\Omega(\frac{\log(n)}{\log(\log(n))})$.

The proof of the lower bound is done by showing an example, such that any online algorithm executed on this example cannot get a competitive ratio less than $\frac{\log(n)}{\log(\log(n))}$. Therefore, the best competitive ratio achieved by any online algorithm is $\frac{\log(n)}{\log(\log(n))}$.

Moreover, the performance we consider is the worst-case performance, so one counter-example is sufficient.

Theorem 3. *Under our settings, the best competitive ratio achieved by any randomized online algorithm against an oblivious adversary is lower bounded by* $\Omega(\frac{\log(n)}{\log(\log(n))})$.

Proof. We prove the theorem through an example. Let $\mathcal{T}$ be a complete binary tree of height H such that:

- Each vertex represents a base station.
- Each edge between two vertices represent the UA cost between the corresponding directly-connected base stations.
- The UA cost from the root to each of its children is D.
- On every path from the root to a leaf, the cost drops by a factor of m on every step.
- The UA cost from vertex i to vertex k is the aggregated cost of the edges from i to k.
- The caching cost of every non-leaf vertex is set to infinity, while the caching cost of the leaves is set to f.

The height of a vertex is the number of edges on the path to the root. The cost from a vertex of height h to each of its children is $\frac{D}{m^h}$. The tree is shown in Figure 4.2.

For a vertex z, let $\mathcal{T}_z$ denote the subtree rooted at vertex z. We observe the following:

- The cost from a vertex of height h to any vertex in $\mathcal{T}_z$ is at most $\frac{mD}{(m-1)m^h}$, which is the UA cost from vertex z to a leaf in $\mathcal{T}_z$.
- The cost from a vertex z of height h to any vertex not in $\mathcal{T}_z$ is at least $\frac{D}{m^{h-1}}$, which is the cost from a vertex z to its parent.

According to Yao's principle [64], it suffices to show that there is a probability distribution over the request sequence, for which the ratio of the expected cost of any deterministic online algorithm to the expected optimal cost is $\Omega(\frac{\log(n)}{\log(\log(n))})$.

To define an appropriate probability distribution, we divide the request sequence into $H+1$ phases. Phase 0 consists of 1 request located at the root. After the end of phase $h, 0 \leq h \leq H$, if z_h is not a leaf, the adversary selects z_{h+1} uniformly at random and independently between the two children of z_h. Phase $h+1$ consists of m^{h+1} requests located at z_{h+1}.

The total number of requests is at most $\frac{m}{m-1}m^H$, which must not exceed n. The optimal solution is to cache the content at z_H, and each phase h except for the last one, incurs a UA cost of at most $\frac{mD}{m-1}$. Therefore, the optimal total cost is at most $f + H\frac{mD}{m-1}$.

Now, let Alg be any deterministic online algorithm, and let $h, 0 \leq h \leq H-1$, be any phase except for the last one. We fix the adversary's random choices $z_0, \ldots, z_h$ up to the end of phase h, and consider the expected cost paid by Alg for requests and caches not in $\mathcal{T}_{z_{h+1}}$.

If Alg did not cache the content at any cache located in $\mathcal{T}_{z_h}$ at the moment the first requests in z_{h+1} arrives, then the content was cached at a cache located in $\mathcal{T}/\mathcal{T}_{z_h}$ from a previous phase. Therefore, the UA cost for the requests located at $z_h \in \mathcal{T}_{z_h}/\mathcal{T}_{z_{h+1}}$ is at least $\frac{m^h D}{m^{h-1}} = mD$, since these requests has to retrieve the content by going through the parent z_h. Otherwise, since z_{h+1} is selected uniformly at random and independently between z_h's children, then, with a probability of at least $1/2$, there is at least one cache located in $\mathcal{T}_{z_h}/\mathcal{T}_{z_{h+1}}$ that cached the content. Therefore for every fixed choice of

$z_0, \ldots, z_h$, the expected cost paid by Alg for requests and caches not in $\mathcal{T}_{z_{h+1}}$ is at least $\min\{mD, f/2\}$, in addition to the costs for requests and caches not located in $\mathcal{T}_{z_h}$.

Hence, at the beginning of phase $h, 0 \leq h \leq H$, the expected cost paid by Alg for requests and caches not in $\mathcal{T}_{z_h}$ is at least $h \min\{mD, f/2\}$. For the last phase, Alg incurs an additional cost of at least $\min\{mD, f\}$ for requests and caches in $\mathcal{T}_{z_H}$.

For $m = H$ and $f = HD$, the total expected cost of Alg is at least $h \min\{HD, HD/2\} + min\{HD, HD\} \leq h\frac{HD}{2} + HD = HD\frac{h+2}{2}$, while the optimal cost is at most $HD\frac{2H-1}{H-1}$. Hence the competitive ratio is lower bounded by $\Omega(H)$. We also have the constraint that the total number of requests, which is at most $\frac{H^{H+1}}{H-1}$ must not exceed n. Setting $H = \lfloor \frac{\log(n)}{\log\log(n)} \rfloor$ yields the claimed lower bound. □

4.5 Simulation Results

4.5.1 Settings

In this section, we compare three caching schemes: the online collaborative scheme (OCC) described in Section 4.4, the optimal offline collaborative caching scheme represented by the ILP formulation described in Section 4.3 and computed using CPLEX [65], and the optimal offline non-collaborative caching scheme, which does not allow collaboration among the base stations, and can be formulated in a similar way to our ILP formulation.

The simulations are run on a random topology, where the base stations are uniformly distributed in a square area of size 50×50 square kilometers and there is a link between two base stations if the distance between them is less than a certain threshold. If we associate a UA cost between any two directly connected base stations, then for

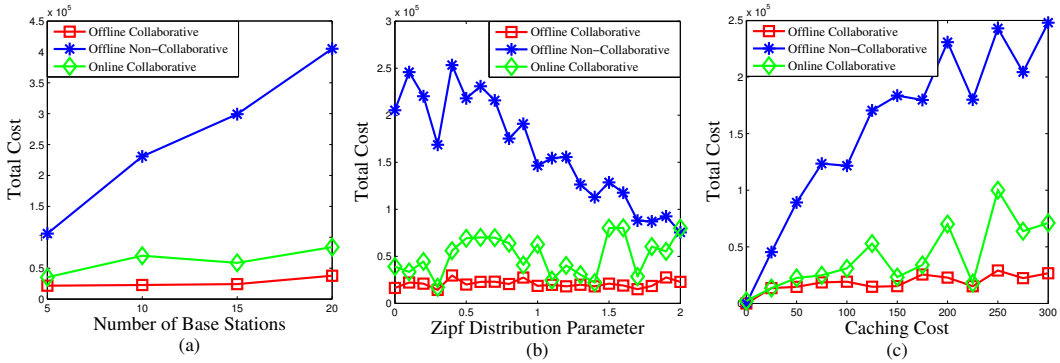

Figure 4.3 Total cost of all schemes.

any two base stations i, k, T_{ij}^{k} can be computed using the path with the minimum UA cost between i and k. We set the number of contents M to 20, where each content has a size chosen uniformly at random from the set $\{10, 11, \ldots, 20\}$MB. The popularity of each content at each base station is chosen according to a Zipf distribution [66], with parameter ζ, where the popularity of a content of rank j is given as $\frac{1/j^{\zeta}}{\sum_{m=1}^{M} 1/m^{\zeta}}$. We assume that content ranking in a base station is different and independent from the ranking in other base stations (*i.e.*, content j may be ranked first in a base station but ranked fifth in another base station). The results in all of the figures are the average of 100 runs.

4.5.2 Results with Accurate Estimation of Content Popularities

We study the effect of changing different parameters on the cost of all schemes. In Figure 4.3(a), we study the effect of changing the number of base stations. As can be observed from this figure, the cost of the non-collaborative scheme is at least 4 times and 3 times that of the cost of the offline and the online collaborative schemes, respectively. We also observe that the offline non-collaborative scheme cost increases at a higher rate than the collaborative schemes when we increase the number of base

stations. The reason behind the above observations is that a base station in the non-collaborative scheme has to retrieve the content from the Internet, if the base station did not cache the content. Alternatively, the base station in the collaborative scheme can retrieve the content from a nearby base station that has the requested content. This shows the scalability of the collaborative caching schemes. Therefore, it is crucial to enable collaboration among the base stations, which is not done in most of the previous work.

In Figure 4.3(b), we study the effect of changing the Zipf distribution parameter ζ. First, we observe that the cost of the non-collaborative scheme is at least 3 times the cost of the offline collaborative scheme, and varies between 1 to 3 times the cost of the online collaborative scheme, as the non-collaborative scheme cannot retrieve a content from a nearby base station. Second, we observe that the offline non-collaborative scheme cost decreases as ζ increases. This is because as ζ increases, less number of contents are requested more often, and the non-collaborative scheme caches the most popular files. Third, we observe that the offline collaborative scheme cost does not change much as ζ increases. This is because the offline collaborative scheme can retrieve contents from nearby base stations.

In Figure 4.3(c), we study the effect of changing the average caching cost. First, we observe that the cost of the non-collaborative scheme is at least 2 times the cost of both of the offline and the online collaborative schemes. Second, we observe that the total cost of the non-collaborative scheme increases at a higher rate than the collaborative schemes as the average caching cost increases. The reason behind both observations is that the non-collaborative scheme does not allow content retrieval from

nearby base stations, which means that each content may be cached at more than one base station, which increases the total caching cost. Contrariwise, the collaborative schemes tend to cache each content at a single base station, and all other base stations can retrieve the content without caching an additional copy of the content or retrieve it from the Internet.

Moreover, in Figure 4.3(c), we start with a caching cost that is less than the attrition cost (*i.e.*, $f_{kj} < T_{ij}^{k}$), then we increase the caching cost until it becomes larger than the attrition cost (*i.e.*, $f_{kj} > T_{ij}^{k}$). We can see a tradeoff between the caching cost and the attrition cost in this figure for the non-collaborative caching scheme. In the non-collaborative scheme, the content is either cached or retrieved from the Internet. On the one hand when the caching cost is low, all base stations tend to cache the content and thus achieving low cost. As the caching cost increases, the base stations tend to retrieve the contents from the Internet instead of caching. However, retrieving from the Internet is costly. On the other hand, when the caching cost is large, the collaborative caching scheme tends to retrieve from nearby base stations. Thus the collaborative caching scheme achieves lower cost than the non-collaborative caching scheme.

From all of the plots in Figure 4.3, we note that enabling collaboration among base stations has a significant impact on the cost reduction. We also note that the cost of the online collaborative scheme is very close to the cost of the optimal offline collaborative scheme, with a maximum degradation of three folds, and that the online collaborative scheme can achieve a cost reduction of four folds over the cost of the offline non-collaborative scheme.

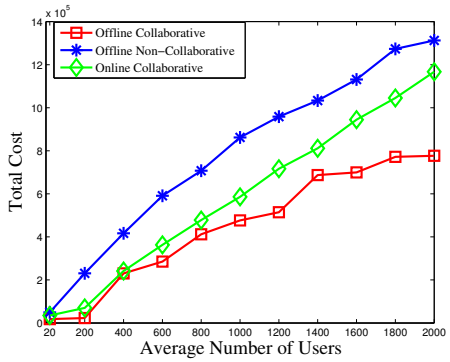

Figure 4.4 Total cost of all schemes vs average number of users.

In Figure 4.4, we study the impact of increasing the average number of users at each base station on the cost of each scheme. As can be seen from this figure, the cost of the online collaborative scheme is less than that of the cost of the non-collaborative caching scheme. This is due to the collaborative property of the online collaborative scheme, where a content can be retrieved from a nearby base station.

In Figure 4.5, we measure the per demand cost savings percentage of the collaborative schemes with respect to the non-collaborative scheme. Here, we measure the cost of the collaborative schemes for 100 different sets of demands. For each set of demand, we normalize the cost of the collaborative schemes with respect to the cost of the non-collaborative scheme, and then subtract it from 1. Denote the cost of the offline collaborative and the offline non-collaborative for the r-th set of demands as $\mathcal{C}_{col}(r)$ and $\mathcal{C}_{non}(r)$, respectively. We compute the per demand cost savings as $R_{col}(r) = (1 - \frac{\mathcal{C}_{col}(r)}{\mathcal{C}_{non}(r)}) \times 100\%$. After that, the empirical CDF of the vector $[R_{col}(1), R_{col}(2), \ldots, R_{col}(100)]$ for the 100 sets of demands is plotted. We do the same process for the online collaborative scheme. From this figure, we observe that the relative cost savings between the offline collaborative scheme and the online

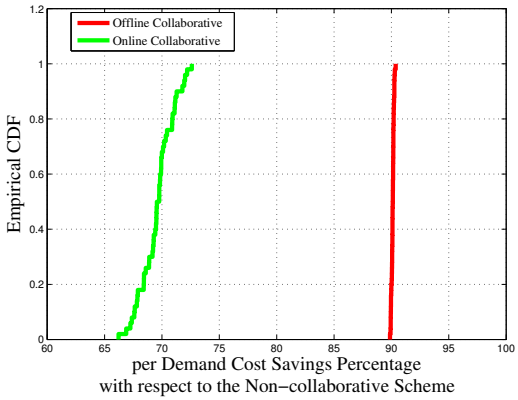

Figure 4.5 Empirical CDF of the per demand cost savings percentage with respect to the non-collaborative scheme.

collaborative scheme is similar among all sets of demands. We also observe that the online collaborative cost savings varies between 65% to 75%. The reason is that the non-collaborative scheme cannot retrieve cached contents from another base station.

4.5.3 Results with Errors in Estimating the Contents Popularities

In Figure 4.6, we repeat the simulations as in Figure 4.3, when a 50% error margin is introduced to the popularity estimation. Formally speaking, we generate two sets of requests, the estimated requests set and the actual requests set, where the estimated requests $\hat{\gamma}_{ij}$ are chosen randomly from a uniform distribution in the range $[0.5\gamma_{ij}, 1.5\gamma_{ij}]$. We use the set of estimated requests to solve the collaborative and the non-collaborative caching optimization problems, then we calculate the total cost based on the answer of the optimization problem and the set of actual requests. In the online collaborative scheme, the total cost is calculated using the actual requests, since the online collaborative scheme works when a new request for a content arrives at a base station.

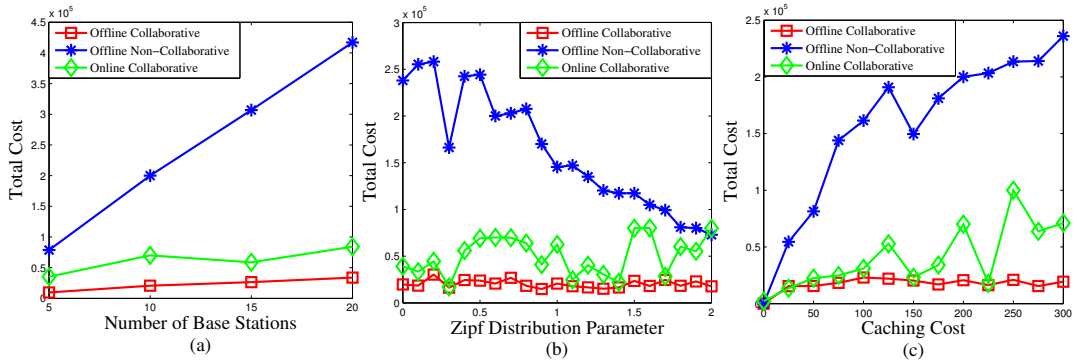

Figure 4.6 Total cost of all schemes with 50% error margin in popularity estimation.

In Figure 4.6(a), we study the effect of changing the number of base stations. As can be observed from this figure, the offline and online collaborative schemes can achieve a cost reduction of at least 500% and 100% over the cost of the non-collaborative scheme, respectively. We also observe that the offline non-collaborative scheme cost grows at a higher rate than the collaborative schemes as the number of base stations increases. The reason is that an error in estimating the contents popularities may cause the non-collaborative scheme not to cache the correct contents, in which case the contents are retrieved from the Internet. Alternatively, the collaborative scheme can retrieve the contents from a nearby base station that has the requested contents.

In Figure 4.6(b), we study the effect of changing the Zipf distribution parameter ζ. First, we observe that the cost of the non-collaborative scheme is increased by at least 3 folds compared to the cost of the offline collaborative scheme, and the cost of the non-collaborative scheme can increase up to 10 folds compared to the cost of the online collaborative scheme. This is because the non-collaborative scheme cannot retrieve a content from a nearby base station. Second, we observe that the offline non-collaborative scheme cost decreases as ζ increases. This is because as ζ increases, less

number of contents are requested more often, and the non-collaborative scheme caches the most popular files. Last, we observe that the offline collaborative scheme cost does not change much as ζ increases. This is because the offline collaborative scheme can retrieve contents from nearby base stations.

In Figure 4.6(c), we study the effect of changing the average caching cost. First, we observe that the cost of the non-collaborative scheme is at least 3 times the cost of both of the offline and the online collaborative schemes. Second, we observe that the total cost of the non-collaborative scheme increases at a higher rate than the collaborative schemes as the caching cost increases. The reason behind both observations is that the non-collaborative scheme prohibits retrieving the contents from nearby base stations, which means that each content may be cached at more than one base station, which increases the total caching cost. Conversely, the collaborative schemes tend to cache each content at a single base station, and all other base stations can retrieve the content without caching an additional copy of the content or retrieve it from the Internet.

In Figure 4.7, we repeat the simulation as in Figure 4.5, when a 50% error margin is introduced to the popularity estimation. We observe that due to the error margin in popularity estimation, there is a small variance in the cost savings of the offline collaborative scheme over the offline non-collaborative scheme. We also observe that the online collaborative scheme cost savings varies between 71% to 76% over the cost of the non-collaborative scheme in all sets of demands.

Figure 4.8 shows the per demand cost savings percentage of the online collaborative schemes with respect to the offline collaborative scheme when the number of base

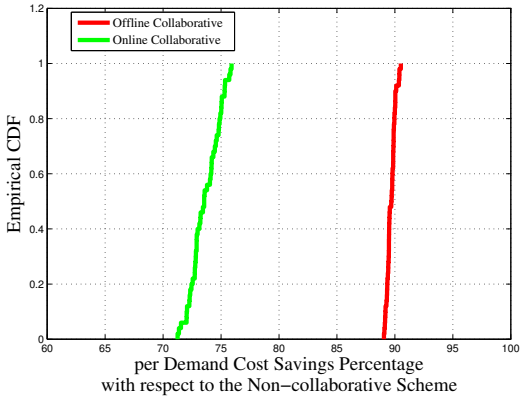

Figure 4.7 Empirical CDF of the per demand cost savings percentage with respect to the non-collaborative scheme with 50% error margin in popularity estimation.

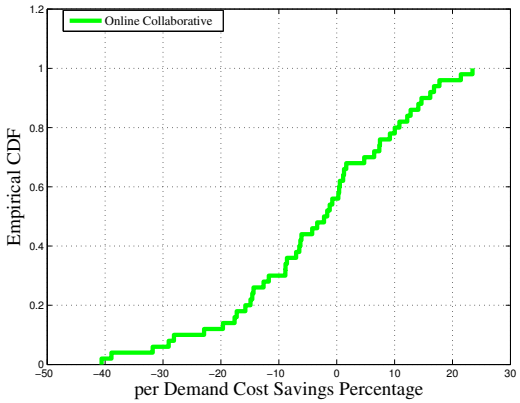

Figure 4.8 Empirical CDF of the per demand cost savings percentage of the online collaborative scheme with respect to the offline collaborative scheme with 50% error margin in popularity estimation.

stations is set to 10, the Zipf distribution parameter ζ is set to 1.1, the average caching cost is set to 200, and a 50% error margin in popularity estimation is introduced. The per demand cost savings percentage is calculated in a similar manner to that in Figure 4.5. From this figure, we note that the online collaborative scheme can outperform the offline collaborative scheme in 40% of the demand sets, and it can achieve around 22% of cost savings over the offline collaborative scheme. This is because the offline collaborative scheme may consider an unpopular content to be popular due to the inaccuracies in estimating the contents popularities, while the online collaborative scheme is not affected by the inaccuracies in estimating content popularities, as it makes a caching decision when a new request for a content arrives at a base station.

In Figure 4.9, we repeat the simulations as in Figure 4.3 when we introduce errors in estimating the contents ranking. The results are averaged over 20 runs. In Zipf popularity distribution, lower ranking number means higher popularity. To introduce the errors in estimating contents ranking, we generate two sets of requests, the estimated requests set and the actual requests set. The estimated requests set is generated by changing the contents ranking at each base station randomly. For example, if the actual ranking of a content at a base station is fifth, then the estimated ranking of the same content at the same base station may be first.

From this figure, we note that the total cost of the non-collaborative scheme varies between 2 times to 9 times of the total cost of the collaborative schemes as the number of base station increases (Figure 4.9(a)). As we change the Zipf distribution parameter, the total cost of the non-collaborative scheme is at least twice the total cost of the collaborative schemes (Figure 4.9(b)). Finally, as the average caching cost increases,

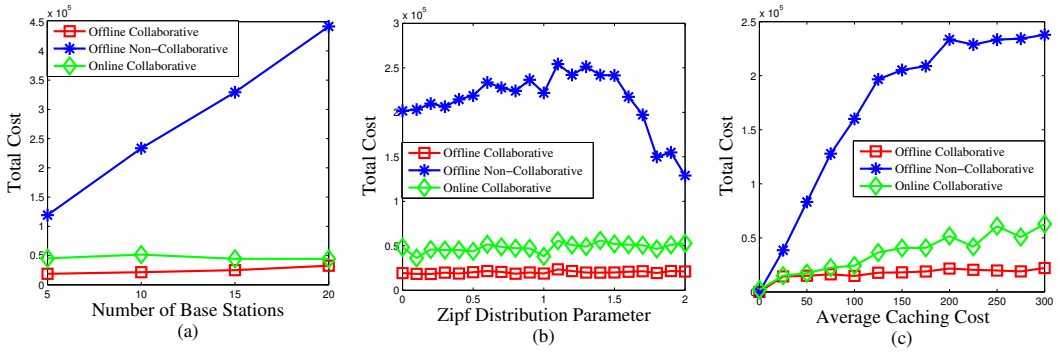

Figure 4.9 Total cost of all schemes with errors in content ranking estimation.

the total cost of the non-collaborative scheme varies between 100% to 400% of the total cost of the collaborative schemes (Figure 4.9(c)). The reason behind all these observations is that the collaborative schemes have a lower total UA cost since the collaborative schemes can retrieve the content from a nearby base station.

Figure 4.10 is similar to Figure 4.7 when we introduce errors in estimating the contents ranking. The results are for 20 different sets of demands. From Figure 4.10, we note that the relative cost savings of the offline collaborative scheme varies between 87% to 92% over all sets of demands, while the relative cost savings of the online collaborative scheme varies between 62% to 92% over all sets of demands.

In Figure 4.11, we measure the relative cost savings of the online collaborative scheme to the offline collaborative scheme over 20 different sets of demands, when we introduce errors in estimating the contents ranking. From this figure, we note that in 30% of the demand sets, the total cost of the online collaborative scheme is less than the total cost of the offline collaborative scheme. This is mainly due to the inaccuracies in estimating the contents ranking.

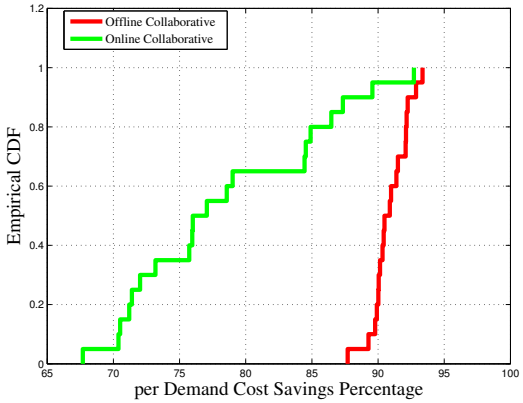

Figure 4.10 Empirical CDF of the per demand cost savings percentage of the online collaborative scheme with respect to the offline collaborative scheme with errors in content ranking estimation.

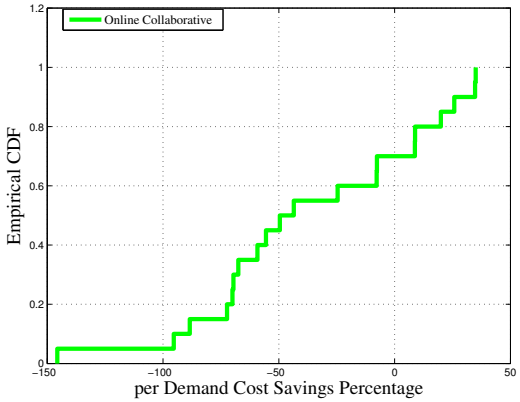

Figure 4.11 Empirical CDF of the per demand cost savings percentage of the online collaborative scheme with respect to the offline collaborative scheme with errors in content ranking estimation.

4.6 Conclusion

In this chapter, we study the problem of content caching in a collaborative multi-cell coordinated system, with the objective of minimizing the total costs paid by the content providers. We formulate the problem of collaborative caching as an optimization problem, and we prove that it is NP-complete. We also provide an online algorithm for the problem. The online algorithm does not require any knowledge about the content popularities. Through extensive simulations, we show that the collaborative caching schemes provide higher savings than the non-collaborative caching scheme, which means that applying the simple online algorithm is better than solving the non-collaborative optimization problem. The simulations also show that our online caching scheme can also outperform the optimal offline collaborative scheme when there are inaccuracies in estimating the contents popularities.

CHAPTER 5

ONLINE ALGORITHM FOR CACHING IN CONTENT CENTRIC NETWORKS

5.1 Introduction

With the proliferation of multimedia content (video, audio, images, ...) generation and sharing, the Internet is becoming more into a content distribution system. As the Internet architecture is based on a host-to-host communication, it is not suitable for content distribution. Recently, Content Centric Networking (CCN), a new paradigm for content delivery in the Internet, has been proposed [16].

Since Internet users are interested in the content itself and not its location, routing in CCN is based on the content's name instead of the IP address of the content's source, which means that a content can be retrieved from nodes that are not on the path to the content's origin server. Content items are given hierarchical names that are understood by the intermediate nodes, and the nodes apply longest prefix match on the content's name for routing decisions [16]. More importantly, nodes in CCN are deployed with storage capabilities that allow the nodes to cache popular content items in order to satisfy future requests. This leads to bandwidth savings as well as latency reduction [16]. Therefore, adopting a well-designed caching policy has a big impact on the performance of CCN. In this chapter[1], we consider content caching in Content Centric Networks (CCN).

[1]The work of this chapter has been published in [67]

We consider caching in CCN from an economical point of view. Our objective is to minimize the overall cost paid by the Content Provider (CP) by introducing a realistic economical model for the Internet Service Provider (ISP) and the CP. In this model, retrieving the content incurs two costs. The first type of cost is the caching cost, where the CP has to pay to the ISP in exchange for caching its content items. This is motivated by the increasing trend of using in-network cloudlets, services, and middleboxes, in which the storage and computations are performed at small clouds installed in the nodes of the network [57–60]. Note that previous work on CCN caching do not consider caching costs, and thus do not provide incentives for the ISP to cache. By introducing caching costs, we consider a realistic model that provides incentives for the ISP to cache.

The second type of cost is what we call the retrieval cost. This cost represents the cost of retrieving the content from other nodes. The retrieval cost can be either a user attrition cost (*i.e.*, the expected cost of losing users to other CPs), or the cost of using the links' bandwidth. These two types of costs yield a tradeoff on where the content items are cached in order to minimize the total cost paid by the CP.

5.2 Settings

We consider a network consisting of $\mathcal{I} = \{1, 2, \ldots, i, \ldots, I\}$ of cache-capable nodes. In the rest of the chapter, we use the words node and cache interchangeably. We have $\mathcal{J} = \{1, 2, \ldots, j, \ldots, J\}$ content items with sizes $\mathcal{S} = \{s_1, s_2, \ldots, s_j, \ldots, s_J\}$ that can be requested by $\mathcal{K} = \{1, 2, \ldots, k, \ldots, K\}$ users.

We assume that the cache size of each node is very large, and instead of imposing cache capacity constraints, we associate a cost f_{ij} with caching the j-th content at the i-th node. Due to the fast development and cost reduction of storage devices, the cache size can be very large with low cost. Moreover, having a caching cost will limit the number of content items cached at a node, and the cache capacity will not be violated.

Let T_{ij}^k denote the retrieval cost associated with retrieving the j-th content for the k-th user from the i-th node. If we associate a cost between any two directly connected nodes, then T_{ij}^k can be computed using the minimum cost path between the i-th node and the k-th user.

5.3 Online Algorithm

In the online version of the problem, the decision of whether to cache a content or not is made when the content is requested. The online algorithm has to make a decision of whether to cache the content at a node or not, and from which node the content is retrieved in order to satisfy the request.

In this section, we present our online algorithm for caching in CCN (OC^3N). The algorithm takes advantage of inherent CCN features such as broadcasting of requests for content items. The difference is that the objective of OC^3N is to minimize the total cost paid by the content provider. The algorithm is lightweight in that it can be implemented with small overhead. Moreover, OC^3N is implemented in a *distributed way, and does not require the knowledge of the network topology or content items' popularities*. The algorithm works as follows:

1. When a request v^j for the j-th content is generated by the k-th user, the user sends the request to next-hop nodes, with initial Time-to-Live (TTL) value.

2. At the node that received the request, if the content is not found and TTL value is greater than 0, the node decreases the TTL value by 1, and sends the request to next-hop nodes. The process continues until the content is found or TTL value reaches 0.

3. If v^j is the first request for the j-th content received by the i-th node, the i-th node initializes a potential function $p(i,j) = 0$. Otherwise, $p(i,j) = \sum_{v' \in V_i^j} [d(W_{v'}^j, v') - d(i,v')]^+$, where V_i^j is the set of requests for the j-th content received by the i-th node excluding the current request v^j, $d(i,v') = T_{ij}^{k'}$ where k'-th user has generated request v', $W_{v'}^j$ is the set of nodes caching the j-th content when request v' is satisfied, $d(W_{v'}^j, v') \equiv \min_{i' \in W_{v'}^j} d(i', v')$, and $[x]^+ \triangleq \max\{x, 0\}$.

4. If TTL value reaches 0 and the content is not found, feedback messages are sent back to the k-th user indicating that the content is not found, along with the current value of $p(i,j)$, the caching cost f_{ij}, and retrieval cost T_{ij}^k of each node i on the path of the feedback messages. The k-th user then increases the initial TTL value and resend the request. The process is repeated until the content item is found.

5. When the content item is found (possibly at multiple nodes), the feedback messages indicate where the content item was found, along with the current values of $p(i,j)$, f_{ij}, and T_{ij}^k of each node i on the path of the feedback messages that did not relay their corresponding values to the k-th user in previous feedback messages before the k-th user increased the TTL value.

6. The k-th user identifies the node $w \leftarrow \arg\max_i (p(i,j) + [d(W_v^j, v^j) - d(i, v^j)]^+ - f_{ij})$.

7. If $p(w,j) + [d(W_v^j, v^j) - d(w, v^j)]^+ - f_{wj} \leq 0$, the k-th user decides not to cache the content at a new node, and sends the value of $d(W_v^j, v^j)$ to all nodes in order for each node i to update its potential function according to $p(i,j) = p(i,j) + [d(W_v^j, v^j) - d(i, v^j)]^+$. Otherwise, the k-th user sends a message to all nodes indicating that node w will cache the content, the node $w' \leftarrow \operatorname{argmin}_{i' \in W_v^j} d(i', v)$ from which node w should retrieve the content, and the value of $d(W_v^j \cup \{w\}, v^j)$ in order for each node to recompute its potential function.

8. Node w then retrieves the content from w' and caches a copy of the content.

9. The k-th user then retrieves the j-th content from the node caching the j-th content and from which the retrieval cost is minimum.

The OC^3N algorithm is presented in Algorithm 7

Algorithm 7 Online Algorithm for Caching in CCN (OC^3N)

$V_i^j \leftarrow \phi, Cost(j) = 0, p(i,j) = 0, \forall i \in \mathcal{I}, j \in \mathcal{J}$
for each new request v^j for the j-th content generated by the k-th user **do**
  Start Exploration Phase
  $t_{v^j} \leftarrow$ initial TTL
  $W_v^j \leftarrow \phi$
  while $W_v^j = \phi$ **do**
    broadcast request v^j to all nodes within t_{v^j} hops from k-th user
    $W_v^j \leftarrow$ set of nodes caching the j-th content
    if $W_v^j = \phi$ **then**
      increase t_{v^j}
  End Exploration Phase
  for all nodes i within t_{v^j} hops from v^j **do**
    $V_i^j \leftarrow V_i^j \cup \{v^j\}$
    updatePotentials(W_v^j, v^j){
    $p(i,j) = p(i,j) + [d(W_v^j, v^j) - d(i, v^j)]^+$
  $w \leftarrow \arg\max_i (p(i,j) - f_{ij})$
  if $p(w,j) - f_{wj} > 0$ **then**
    $w' \leftarrow \text{argmin}_{i' \in W_v^j} d(i', v)$
    $Cost(j) = Cost(j) + f_{wj} + d(w', w)$
    $W_v^j \leftarrow W_v^j \cup \{w\}$
    for all nodes i within t_{v^j} hops from v^j **do**
      computeNewPotentials(V_i^j, w){
      $p(i,j) = \sum_{v' \in V_i^j} [d(W_{v'}^j \cup \{w\}, v') - d(i, v')]^+$
  Start Assignment Phase
  assign v^j to $\alpha = \text{argmin}_{i' \in W_v^j} d(i', v^j)$
  $Cost(j) = Cost(j) + d(\alpha, v^j)$
  End Assignment Phase

We note that there is an overhead required when executing the OC^3N algorithm (we explain the required overhead in details in Section 5.5). However, this overhead is independent of the content's size. Therefore, as the size of the content becomes larger, which is the case for future content delivery traffic, the overhead will not grow.

5.4 Performance Analysis

Before we present the performance analysis, we point out the following observation. The decision of caching a content at a node and updating the potential function for a content is independent from the other content items. so we can view our problem as J independent caching subproblems. In the sequel, we only consider a single content, and hence, the content-index j is omitted in the subscript of the symbols used throughout the rest of the chapter. Moreover, in the online algorithm and in the proof of the competitive ratio, every term will be multiplied by the content size s_j. For simplicity, we set $s_j = 1$. This will not affect the final result of the proof.

To characterize the competitive ratio achievement of the algorithm, we compare the algorithm's cost with the cost of the optimal offline solution. In the optimal offline solution, let W^* denote the set of nodes caching the content. Then, the cost of the offline optimal solution is given by:

$$\mathcal{C}^* = \sum_{w \in W^*} f_w + \sum_{v \in V} d(W^*, v) \tag{5.1}$$

5.4.1 Proof Outline

Let the optimal offline solution W^* consist of L caches $c_1, c_2, \ldots, c_l, \ldots, c_L$. In the optimal offline solution, each request is satisfied by retrieving the content from a

cache. Hence, W^* divides the requests into optimal clusters $C_1, C_2, \ldots, C_l, \ldots, C_L$. For example, if the optimal solution decides to cache a content in three caches c_1, c_2, and c_3, then the first cluster C_1 consists of all requests retrieving the content from c_1, the second cluster C_2 consists of all requests retrieving the content from c_2, and so on.

We start by proving that OC^3N maintains the invariant $p(i) \leq f_i$ for all i (Lemma 9). Based on Lemma 9 and the triangular inequality, we show that the sum of the costs between the optimal cache c_l and the online cache in W_v with the minimum $d(W_v, v)$ for all requests $v \in C_l$ is less than $f_{c_l} + 2\sum_{v \in C_l} d(c_l, v)$. This is used to show that the retrieval cost incurred by OC^3N is within a constant factor of the total optimal cost $\mathcal{C}^*$ (Lemma 10).

For each new request v, we define a credit $\hat{c}(v) = \min\{d(W_v, v), \min_i\{f_i - p(i) + d(i, v)\}\}$. We show that $\hat{c}(v) = f_w - p(w) + d(w, v)$, if v causes the content to be cached at w, and $\hat{c}(v) = d(W_v, v)$ otherwise (Lemma 11). We then show that the total caching cost incurred by OC^3N is bounded by the total credit for all requests, which in turn is within a constant factor of the optimal offline cost $\mathcal{C}^*$. (Lemma 12).

Lastly, using Lemma 10, Lemma 12 and the triangular inequality, we show that if the algorithm decides to cache at a new cache w, then the cost of transferring the content from the old cache to the new cache is within a constant factor of the total optimal cost $\mathcal{C}^*$ (Lemma 13). By combining the bounds from Lemma 10, Lemma 12, and Lemma 13, we show that the competitive ratio of OC^3N is $\mathcal{O}(1)$.

5.4.2 The Proof

Lemma 9. $p(i) \leq f_i, \forall i \in \mathcal{I}$.

Proof. We prove this lemma by induction on the number of requests considered by the online algorithm. For the first request, the invariant holds since $p(i) = 0$ for all $i \in \mathcal{I}$. We inductively assume that the invariant holds just before a new request v arrives and prove that the invariant holds after v is assigned to retrieve the content from a cache.

Let W_v be the set of caches caching the content found after the exploration phase in OC^3N, and let V be the set of requests considered so far by OC^3N. By definition, $p(i) = \sum_{v' \in V_i} [d(W_{v'}, v') - d(i, v')]^+$ is the potential of cache i just before the new request v arrives, and by the induction hypothesis, $p(i) \leq f_i, \forall i$. Let $p'(i)$ be the potential after the subroutine *updatePotentials()* in OC^3N is executed. Finally, let $p''(i)$ be the potential of cache i after the request v is assigned to a cache. What remains to prove is that $p''(i) \leq f_i$ for all i.

We have two cases:

- First case: If the new request v does not change W_v (*i.e.*, the new request did not cause the content to be cached at a new cache), then from OC^3N, $p'(i) - f_i \leq 0, \forall i$, and $p''(i) = p'(i)$ since the subroutine *computeNewPotentials()* is not executed. Therefore, $p''(i) \leq f_i$ for all i.
- Second case: If the new request v causes the content to be cached at a new cache w, then

$$\begin{aligned} 0 < p'(w) - f_w &= [d(W_v, v) - d(w, v)]^+ + p(w) - f_w \\ &\leq [d(W_v, v) - d(w, v)]^+ \end{aligned}$$

  where the first inequality holds because the content is cached at w, the next equality holds from the definition of $p'(w)$, and the last inequality holds from the induction hypothesis $p(w) \leq f_w$. Therefore,

$$[d(W_v, v) - d(w, v)]^+ \geq p'(w) - f_w > 0 \tag{5.2}$$

  This implies that $d(W_v, v) > d(w, v)$, which means that v will retrieve the content from w and

$$d(W_v \cup \{w\}, v) = d(w, v) \tag{5.3}$$

  From here, we have two subcases:

– First subcase: For all caches i where $d(i,v) < d(w,v)$, we have $[d(W_v \cup \{w\}, v) - d(i,v)]^+ = d(w,v) - d(i,v)$. Therefore,

$$\begin{aligned} d(W_v, v) - d(w, v) & \\ &\geq p'(w) - f_w \\ &\geq p'(i) - f_i \\ &= [d(W_v, v) - d(i, v)]^+ + p(i) - f_i \\ &\geq d(W_v, v) - d(i, v) + p(i) - f_i \end{aligned}$$

where the first inequality follows from (5.2), the second inequality follows since $w \leftarrow \arg\max_i(p'(i) - f_i)$, and the first equality follows from the definition of $p'(i)$. Therefore, $d(W_v, v) - d(w, v) \geq d(W_v, v) - d(i, v) + p(i) - f_i$. Rearranging the terms we get

$$\begin{aligned} 0 \geq & d(w, v) - d(i, v) + p(i) - f_i \\ \geq & d(w, v) - d(i, v) + \sum_{v' \in V_i} [d(W_{v'}, v') - d(i, v')]^+ - f_i \\ \geq & d(w, v) - d(i, v) \\ & + \sum_{v' \in V_i} [d(W_{v'} \cup \{w\}, v') - d(i, v')]^+ - f_i \\ = & p''(i) - f_i \end{aligned}$$

where the second inequality follows from the definition of $p(i)$, the third inequality follows from the fact that $d(W_{v'}, v') \geq d(W_{v'} \cup \{w\}, v')$ for all v', and the last inequality follows from the definition of $p''(i)$ and using (5.3).

– Second subcase: For all caches k where $d(i,v) \geq d(w,v)$, we have $[d(W_v \cup \{w\}, v) - d(i,v)]^+ = 0$. Therefore,

$$\begin{aligned} p''(i) &= \sum_{v' \in V_i \cup \{v\}} [d(W_{v'} \cup \{w\}, v') - d(i, v')]^+ \\ &= \sum_{v' \in V_i} [d(W_{v'} \cup \{w\}, v') - d(i, v')]^+ \\ &\leq \sum_{v' \in V_i} [d(W_{v'}, v') - d(i, v')]^+ \\ &= p(i) \leq f_i \end{aligned}$$

where the first equality holds from the definition of $p''(i)$, the second equality follows since $[d(W_v \cup \{w\}, v) - d(i,v)]^+ = 0$, the third inequality follows since $d(W_{v'} \cup \{w\}, v') \leq d(W_{v'}, v'), \forall v'$, and the last equality follows from the definition of $p(i)$ before the request v appears.

□

In the next lemma, we bound the retrieval cost incurred by OC^3N.

Lemma 10. *Let V be the set of requests, and let $\sum_{v\in V} d(W_v, v)$ denote the total retrieval cost incurred by OC^3N. Then*

$$\sum_{v\in V} d(W_v, v) \leq 3\mathcal{C}^*$$

Proof. Let C_l be an optimal cluster with cache c_l. For each request $\hat{v} \in C_l$, let $W_{\hat{v}}$ be the set of caches caching the content at the end of the assignment phase of request $\hat{v}$ in OC^3N. Then, using triangular inequality we have

$$d(W_{\hat{v}}, \hat{v}) \leq d(W_{\hat{v}}, c_l) + d(c_l, \hat{v})$$

Summing over all $\hat{v} \in C_l$, we get

$$\sum_{\hat{v}\in C_l} d(W_{\hat{v}}, \hat{v}) \leq \sum_{\hat{v}\in C_l} d(W_{\hat{v}}, c_l) + \sum_{\hat{v}\in C_l} d(c_l, \hat{v}) \tag{5.4}$$

Now for the optimal cluster C_l with cache c_l we have:

$$\begin{aligned}
p(c_l) &= \sum_{v\in V_{c_l}} [d(W_v, v) - d(c_l, v)]^+ \\
&\geq \sum_{v\in C_l} [d(W_v, v) - d(c_l, v)] \\
&\geq \sum_{v\in C_l} [d(W_v, c_l) - d(c_l, v) - d(c_l, v)] \\
&\geq \sum_{v\in C_l} d(W_v, c_l) - 2\sum_{v\in C_l} d(c_l, v)
\end{aligned}$$

where the second inequality is obtained by using the triangular inequality (*i.e.*, $d(W_v, v) \geq d(W_v, c_l) - d(c_l, v)$).

Using the invariant $f_{c_l} \geq p(c_l)$ from Lemma 9 and rearranging the terms, we get:

$$\sum_{v \in C_l} d(W_v, c_l) \leq f_{c_l} + 2 \sum_{v \in C_l} d(c_l, v) \tag{5.5}$$

Substituting (5.5) into (5.4) we get

$$\begin{aligned} \sum_{\hat{v} \in C_l} d(W_{\hat{v}}, \hat{v}) &\leq [f_{c_l} + 3 \sum_{v \in C_l} d(c_l, \hat{v})] \\ &\leq 3[f_{c_l} + \sum_{\hat{v} \in C_l} d(c_l, \hat{v})] \end{aligned}$$

The lemma follows by summing over all clusters. □

We bound the total caching cost incurred by OC^3N in the next two lemmas. First, we show that the credit $\hat{c}(v) = f_w - p(w) + d(w, v)$, if v causes the content to be cached at w, and $\hat{c}(v) = d(W_v, v)$ otherwise. Then we show that the total caching cost is bounded by the total credit for all requests, which in turn is within a constant factor of the optimal offline cost $\mathcal{C}^*$.

Lemma 11. *For each new request v, let W_v be the set of caches caching the content seen by request v after the exploration phase in OC^3N. Then $\hat{c}(v) = f_w - p(w) + d(w, v)$ if v causes the content to be cached at w, and $\hat{c}(v) = d(W_v, v)$ otherwise.*

Proof. Let $p(i)$ denote the potential function of the i-th cache just before a new request v arrives, and let $p'(i) = p(i) + [d(W_v, v) - d(i, v)]^+$ be the potential function after the subroutine *updatePotential()* in OC^3N is executed.

Now, if the new request v causes the content to be cached at w, then from the algorithm,

$$\begin{aligned} 0 < p'(w) - f_w &= p(w) + [d(W_v, v) - d(w, v)]^+ - f_w \\ &= p(w) + d(W_v, v) - d(w, v) - f_w \\ &\geq p(i) + [d(W_v, v) - d(i, v)]^+ - f_i \\ &\geq p(i) + d(W_v, v) - d(i, v) - f_i \end{aligned}$$

The first inequality holds because the content is cached at w. The first equality follows from the definition of $p'(w)$. The second equality follows by using (5.3). The second inequality holds because $p'(w) - f_w \geq p'(i) - f_i, \forall i$ and from the definition of $p'(i)$. Therefore, $p(w)+d(W_v, v)-d(w, v)-f_w \geq p(i)+d(W_v, v)-d(i, v)-f_i$. Rearranging the terms we get that $f_w - p(w) + d(w, v) \leq f_i - p(i) + d(i, v)$. We also have from the second inequality that $p(w) + d(W_v, v) - d(w, v) - f_w > 0$. Therefore, $\hat{c}(v) = f_w - p(w) + d(w, v)$ if v causes the content to be cached at w.

On the other hand, if the new request v does not cause an additional copy of the content to be cached, then from the algorithm,

$$\begin{aligned} 0 \geq p'(i) - f_i & \qquad \forall i \\ = p(i) + [d(W_v, v) - d(i, v)]^+ - f_i & \qquad \forall i \\ \geq p(i) + d(W_v, v) - d(i, v) - f_i & \qquad \forall i \end{aligned}$$

where the first equality follows from the definition of $p'(i)$. Therefore, $d(W_v, v) \leq f_i - p(i) + d(i, v), \forall i$, and $\hat{c}(v) = d(W_v, v)$. □

In the next lemma, we show that the total caching cost incurred by OC^3N is upper bounded by the total optimal cost $\mathcal{C}^*$.

Lemma 12. *Let V be the set of requests, and let W_v be the set of caches caching the content seen by request v after the exploration phase in OC^3N. Let $\mathcal{W}$ denote the set of caches caching the content after all the requests in V have been considered and let $\sum_{w\in\mathcal{W}} f_w$ denote the total caching cost incurred by OC^3N. Then*

$$\sum_{w\in\mathcal{W}} f_w \leq 3\mathcal{C}^*$$

Proof. We first show that the total caching cost incurred by OC^3N is upper bounded by the total credit for all requests. We do this by a potential function argument. We define the potential function $\Phi = \sum_{v\in V} d(W_v, v)$ and calculate the change $\Delta\Phi$ in the value of the potential function when a new request v is considered. Let $p(i)$ be the value of the potential function of the i-th cache just before the new request v arrives.

If the new request v does not cause the content to be cached at a new cache (*i.e.*, W_v is not changed), then $\Delta\Phi = d(W_v, v) = \hat{c}(v)$ by Lemma 11. Otherwise, if v causes the content to be cached at w, then $d(W_v \cup \{w\}, v) = d(w, v)$ (recall (5.3) in Lemma 9),

and $d(W_{v'}, v') - d(W_{v'} \cup \{w\}, v') = [d(W_{v'}, v') - d(w, v')]^+$ for all $v' \in V$. Therefore,

$$\begin{aligned}
\Delta\Phi &= \sum_{v' \in V \cup \{v\}} d(W_{v'} \cup \{w\}, v') - \sum_{v' \in V} d(W_{v'}, v') \\
&= d(w, v) - \sum_{v' \in V} [d(W_{v'}, v') - d(W_{v'} \cup \{w\}, v')] \\
&= d(w, v) - \sum_{v' \in V} [d(W_{v'}, v') - d(w, v')]^+ \\
&\leq d(w, v) - \sum_{v' \in V_w} [d(W_{v'}, v') - d(w, v')]^+ \\
&= d(w, v) - p(w)
\end{aligned}$$

where the last equality follows from the definition of $p(w)$. From Lemma 11, we have $\hat{c}(v) = f_w - p(w) + d(w, v) \geq f_w + \Delta\Phi$. Therefore, $\sum_{v \in V} \hat{c}(v) \geq \Phi + \sum_{w \in \mathcal{W}} f_w \geq \sum_{w \in \mathcal{W}} f_w$ since $\Phi \geq 0$.

Now we use (5.5) in Lemma 10 to show that the total credit of the requests in V is within a constant factor of the total optimal cost $\mathcal{C}^*$. Let C_l be an optimal cluster with cache c_l. Let $n_l \equiv |C_l|$ be the number of requests in C_l, and let $v_1, v_2, \ldots, v_n, \ldots, v_{n_l}$ be the requests in C_l in the order considered by OC^3N. For each request $v_n \in C_l$, let W_{v_n} be the set of caches caching the content at the end of the assignment phase of request v_n in OC^3N.

The credit of each request v_n is $\hat{c}(v_n) \leq \min\{d(W_{v_n}, \hat{v}),\ f_{c_l} + d(c_l, v_n)\}$ (using Lemma 11). For the first request, $\hat{c}(v_1) \leq f_{c_l} + d(c_l, v_1)$. For the remaining requests $v_n, n \geq 2$, we have

$$\hat{c}(v_n) \leq d(W_{v_n}, v_n) \leq d(W_{v_n}, c_l) + d(c_l, v_n)$$

where the second inequality follows from the triangular inequality.

Summing over all $v_n \in C_l$, we get

$$\begin{aligned}\hat{c}(v_1) + \sum_{n=2}^{n_l} \hat{c}(v_n) &\leq f_{c_l} + d(c_l, v_1) \\ &\quad + \sum_{n=2}^{n_l} [d(W_{v_n}, c_l) + d(c_l, v_n)] \\ &\leq f_{c_l} + \sum_{n=1}^{n_l} d(W_{v_n}, c_l) + \sum_{n=1}^{n_l} d(c_l, v_n) \\ &\leq 2f_{c_l} + 3\sum_{n=1}^{n_l} d(c_l, v_n) \\ &\leq 3[f_{c_l} + \sum_{n=1}^{n_l} d(c_l, v_n)]\end{aligned}$$

where the third inequality follows by using (5.5) in Lemma 10. Summing over all clusters we get that

$$\sum_{w \in \mathcal{W}} f_w \leq \sum_{v \in V} \hat{c}(v) \leq 3\mathcal{C}^*$$

□

In the next lemma, we use the triangular inequality, Lemma 10 and Lemma 12 to show that the total cost incurred by OC^3N when the content is transferred to a new cache w is within a constant factor of the total optimal cost $\mathcal{C}^*$.

Lemma 13. *In OC^3N, let V denote the set of requests and let $V' \subset V$ denote the subset of requests that caused the content to be cached at a new cache. For all $v \in V'$, let W_v denote the set of caches seen by request v after the exploration phase, and let w_v denote the new cache. Then*

$$\sum_{v \in V'} d(W_v, w_v) \leq 9\mathcal{C}^*$$

Proof. Using the triangular inequality, we have

$$\begin{aligned} d(W_v, w_v) &\leq d(W_v, v) + d(w_v, v) \\ &\leq d(W_v, v) - d(w_v, v) + 2d(w_v, v) \\ &\leq \sum_{v' \in V_{w_v}} [d(W_{v'}, v') - d(w_v, v')]^+ + 2d(w_v, v) \\ &\leq p(w_v) + 2d(w_v, v) \\ &\leq f_{w_v} + 2d(w_v, v) \end{aligned}$$

where the fourth inequality follows from the definition of $p(w_v)$, and the last inequality follows from Lemma 9. Summing over all $v \in V'$ we get

$$\begin{aligned} \sum_{v \in V'} d(W_v, w_v) &\leq \sum_{v \in V'} [f_{w_v} + 2d(w_v, v)] \\ &\leq \sum_{v \in V'} f_{w_v} + 2 \sum_{v \in V} d(w_v, v) \\ &\leq 9\mathcal{C}^* \end{aligned}$$

where the last inequality follows by using Lemma 10 and Lemma 12. □

Now we are ready to prove the competitive ratio of OC^3N.

Proposition 3. *The competitive ratio of OC^3N is $\mathcal{O}(1)$.*

Proof. From Lemma 10, we have $\sum_{v \in V} d(W_v, v) \leq 3\mathcal{C}^*$. From Lemma 12, we have $\sum_{w \in \mathcal{W}} f_w \leq 3\mathcal{C}^*$. Finally, from Lemma 13, we have $\sum_{v \in V'} d(W_v, w_v) \leq 9\mathcal{C}^*$. The proposition follows by combining the three bounds. □

Note that this is the worst case ratio and it does not depend on the problem size, so it is optimal from the asymptotic sense.

5.5 Practical Issues

In this section, we discuss the practical issues concerning the implementation of OC^3N.

5.5.1 Broadcasts

For OC^3N to be implemented, two broadcasts are required. The first broadcast includes the request itself, which is a feature of CCN. The second broadcast required is the broadcast to notify the nodes of whether the content is to be cached at a new node or not, along with the required information for every node to update their potential functions. The required information can be included in the broadcast message as three fields. The first field is a single bit indicating whether the content is cached at a new cache or not, the second field contains the ID of the new cache or is left empty if the content is not cached at a new cache, and the third field contains the value of $d(W_v, v)$ at the beginning of the assignment phase. Note that the size of the second broadcast is very small when compared to the content's size, and therefore adds a very small overhead to the implementation of OC^3N.

5.5.2 Retrieving from Multiple Nodes

In OC^3N, the user first decides whether to cache the content at a new node or not, and then retrieves the content. Due to this implementation, only one copy of the content is sent back to the user. Content retrieval occurs after the second broadcast discussed previously, and this introduces additional delays to the system.

In order to avoid these delays, OC^3N implementation can be changed to retrieve the content after the first broadcast. This may cause multiple copies of the content to be sent back to the user along multiple paths, and each node on the path temporarily

caches the content until it is informed otherwise. Note that this implementation makes OC^3N more compliant to CCN, since the original implementation of CCN can also result in retrieving multiple copies of the content due to having a single broadcast phase. However, this causes the competitive ratio of OC^3N to be $\mathcal{O}(n)$ as stated in the following proposition.

Proposition 4. *The competitive ratio of OC^3N is $\mathcal{O}(n)$ if the content is retrieved from multiple nodes.*

Proof. We show this proposition by an example. In this example, the network is represented by a complete binary tree consisting of n nodes as shown in Figure 5.1. The content is cached at every leaf of the tree and a request for the content arrives at the root of the tree. Since there are $n/2$ leaves, $n/2$ copies will be sent back to the user (one copy will be accepted and the rest will be discarded), and each one of those copies has a retrieval cost that is within a constant factor of the total optimal cost (refer to Lemma 10 in Section 5.4). Therefore, this implementation will have a competitive ratio of $\mathcal{O}(n)$. □

5.5.3 Executing *computeNewPotentials()* Subroutine

In the case where the content is to be cached at a new cache, the nodes are required to recompute their respective potential functions by executing the *computeNewPotentials()* subroutine. In order to do so, the nodes need to maintain a table containing information regarding past requests for the content. Specifically, the i-th node needs to store in its table the values of $d(W_v, v)$ and $d(i, v)$ for every request $v \in V_i$. As the number of requests increases, the size of the table increases. Nevertheless, note that an individual

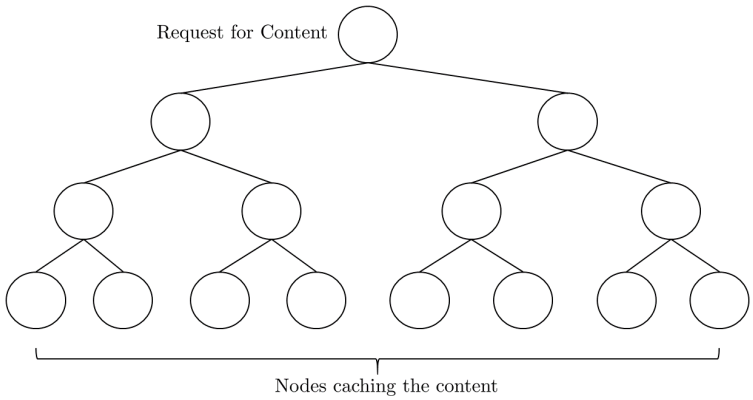

Figure 5.1 Example network used in the proof of Proposition 4.

entry in the table is represented by two numbers. Therefore, the size of an individual entry in the table is very small when compared to the content's size, and since we assume large cache sizes, these information can be easily maintained.

5.5.4 Pricing Model

The pricing model adopted so far for OC^3N is one where the caching cost changes every period of time and the ISP evicts the content items at the end of each period. For the case where the content items are to be cached for a fixed time T (like 24 hours), and the caching cost does not change, we propose a heuristic based on OC^3N, named OC^3N_Fixed. To implement this heuristic, we add an eviction phase to Algorithm 7 after the end of the assignment phase, in order to evict the content items after a fixed time T.

5.6 Simulation Results

In this section, we measure the performance of OC^3N via simulations. The simulations are run on a random topology consisting of 100 nodes uniformly distributed in a square area of size 2000×2000 square kilometers, and there is a link connecting two nodes if the distance between them is less than 400 kilometers. We set the number of content items J to 1000, where each content has a size chosen from the set $\{10, 11, \ldots, 20\}$MB. The popularity of each content is chosen according to a Zipf distribution [66], with parameter $\zeta = 0.8$, where the popularity of a content of rank q is given as $\frac{1/q^{\zeta}}{\sum_{j=1}^{J} 1/j^{\zeta}}$. For the retrieval cost, we assume that the cost to transmit 2.7GB of data over a 100km costs \$1 as adopted from [68], and the retrieval cost value T_i^k between the i-th node and the k-th user is computed using the minimum cost path. For the caching cost, we adopt the pricing from Amazon EC2 [69].

We first compare the performance of OC^3N vs. the performance of two heuristics, Leave Copy Down (LCD) and Leave Copy Everywhere (LCE) [70]. The results are shown in Figure 5.2, where every point is the average of 100 runs.

First, we measure the effect of increasing the number of users on the total cost of all schemes. The results are shown in Figure 5.2(a). As can be seen from this figure, as the number of users increases, the total cost of all schemes increases. This is expected since increasing the number of users translates into more requests for the content. In addition, we note that OC^3N can cut the total cost incurred by the LCD and LCE caching schemes by up to 57% and 65%, respectively. This is because the LCD and LCE caching schemes end up caching the content at all nodes as opposed to OC^3N. Moreover, we note that the total cost of the LCD scheme is greater than the total cost of

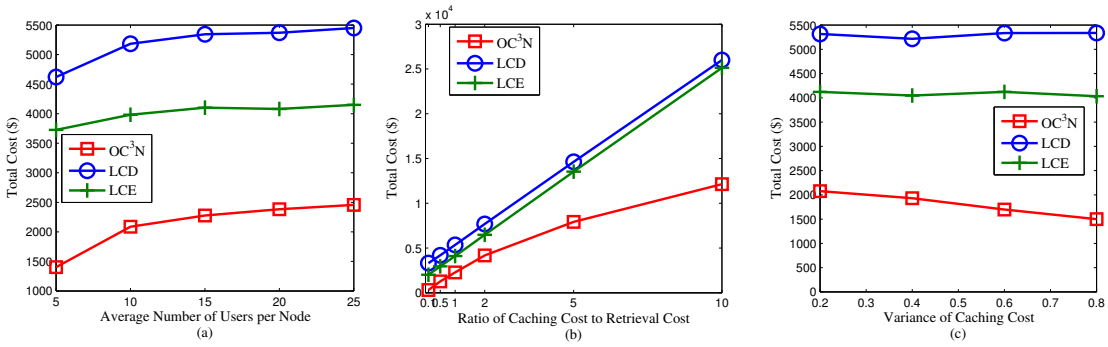

Figure 5.2 Total cost vs. different parameters.

the LCE scheme. This is because LCD scheme incurs more retrieval cost for requests for the same content generated at the same node, since the content is brought closer one hop at a time with each request along the path from the content's source to the user, while LCE scheme caches the content at every node along the path, so future requests generated at the same node will not incur any additional costs.

Next, we measure the total cost of all schemes as we change the ratio of the caching cost to the retrieval cost. This is done by multiplying the caching cost by the desired ratio. The results are shown in Figure 5.2(b). As shown in this figure, as the ratio increases, the total cost of all schemes increases since if the content is cached at a new cache, additional charges are paid by the content provider. We also observe that increasing the ratio of the caching cost to the retrieval cost has bigger impact on the total cost of the LCD and LCE caching schemes than on the total cost of OC^3N, since LCD and LCE caching schemes end up caching the content at every node, while OC^3N will rarely cache the content at a new cache as the caching cost increases.

Lastly, we measure the total cost of all schemes vs. the variance of the caching cost. The caching costs are drawn from a uniform distribution. The results are shown

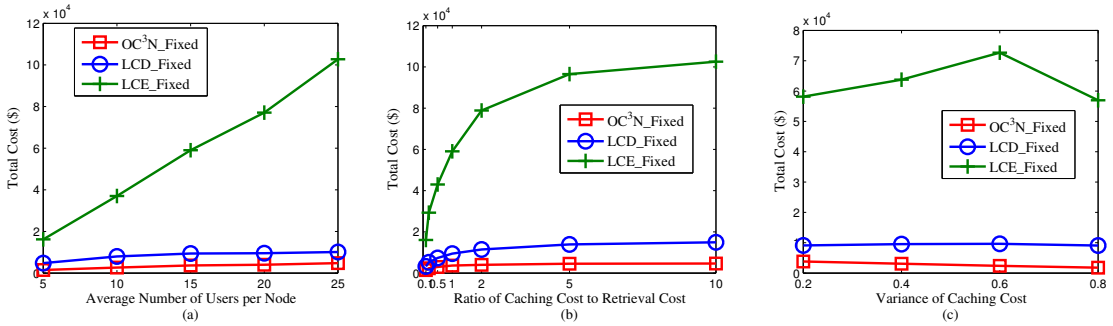

Figure 5.3 Total cost vs. different parameters when content items are cached for a fixed time.

in Figure 5.2(c). We observe from this figure that the caching cost variance does not have an effect on the LCD and LCE schemes, since both schemes will end up caching the content at all nodes. However, having a large variance of the caching cost benefits OC^3N, since the content will be cached at the nodes with low caching costs.

Next, we compare OC^3N_Fixed heuristic proposed in Section 5.5.4 against modified versions of LCD and LCE schemes, where the content items are cached for a fixed time. We repeat the same simulations as in Figure 5.2, and the results are shown in Figure 5.3.

In Figure 5.3(a), we measure the total cost vs. the average number of users. As can be seen from this figure, increasing the number of users has bigger impact on the total cost of the LCE scheme than the total cost of OC^3N_Fixed and LCD schemes , since in LCE scheme, multiple nodes evict the content at the same time, and recaching the content again incurs additional costs. Moreover, OC^3N_Fixed can cut the cost incurred by the LCD scheme by up to 70%, since the LCD scheme will end up caching the content at all nodes, thus increasing the total cost.

In Figure 5.3(b), we measure the total cost of all schemes as we change the ratio of the caching cost to the retrieval cost. Again, the LCE scheme incurs the highest cost,

since in this scheme, multiple nodes evict the content at the same time, and recaching the content again incurs additional costs. Also, as seen from this figure, as the ratio increases, the total cost of OC^3N_Fixed slightly increases, since the content will rarely be cached at a new cache. Lastly, we note that OC^3N_Fixed can cut the cost incurred by the LCD scheme by up to 83%.

Lastly, in Figure 5.3(c), we measure the total cost vs. the variance of the caching cost. The caching costs are drawn from a uniform distribution. We note that the total cost of OC^3N_Fixed is minimum when the variance of the caching cost is maximum. This is because when the variance is large, there are some nodes that have low caching cost, and OC^3N_Fixed tends to cache at those nodes only. Contrariwise, the total cost of the LCD scheme slightly changes, since the LCD scheme ends up caching the content at all nodes.

5.7 Conclusion

In this chapter, we study the problem of caching in Content Centric Networking from an economical point of view, where the content provider is charged in exchange for caching its content items. This is motivated by the increasing trend of using in-network cloudlets, where the nodes become a small cloud. We propose an online algorithm (OC^3N) that minimizes the total costs paid by the content provider. The algorithm works on a per-request basis and does not require the exact knowledge of content items' popularities.

Through detailed analysis, we show that the total cost incurred by OC^3N is within a constant factor of the optimal total cost for the cases where the caching

cost changes periodically, and the content items are evicted at the end of the period. For the cases where the caching cost does not change, we propose a heuristic based on OC^3N where the content items are cached for a fixed time. Moreover, through simulations, we investigate the effect of different parameters on the performance of the proposed algorithms, and show that the proposed algorithms can cut the total cost incurred by widely used caching schemes such as Leave Copy Down (LCD) and Leave Copy Everywhere (LCE) by up to 65%.

CHAPTER 6

SUMMARY AND FUTURE DIRECTIONS

6.1 Summary

Due to the proliferation of content delivery services in the recent years, network operators need to devise new ways in order to enhance their networks' performance. One way to address this challenge is to provide content caching at intermediate nodes. Content caching at intermediate nodes brings the contents closer to the requesting customers. This has the effect of offloading the traffic from the origin servers, reducing content delivery time, and achieving better performance, scalability, and energy efficiency.

In this dissertation, we studied the problem of content caching under a framework that brings incentives for the nodes to cache the contents, where content providers are required to pay the Internet Service Provider (ISP) in exchange of caching their contents, and the ISP in return is required to provide QoS guarantees by not replacing the contents of the content providers in the future, if the ISP decides to cache their contents. We proposed three different online caching algorithms that take the charges paid by the content providers into consideration in order to achieve their respective objectives, where the algorithm decides whether to cache a content or not at the time that content is requested.

First, we presented an online caching algorithm in Chapter 3 for the current Internet network, where a request for a content is forwarded along a single path to the content origin server. With the objective of maximizing the traffic savings, our

proposed online Cost-Reward Caching (CRC) algorithm decides which nodes along this path are to cache the content. Based on a comparison between the value of a cost function that is exponentially proportional to the relative load of the caching node and the expected traffic savings achieved if the content is to be cached, a caching decision is made. The proposed algorithm is easily implemented in the current Internet networks in a distributed way. Our theoretical analysis showed that our algorithm achieves the optimal competitive ratio in the asymptotic sense as the number of caching nodes in the network increases. We also proposed multiple extensions to the CRC algorithm that either focus on minimizing the energy consumption, considers replacement, or both. Our simulation results showed that the CRC algorithm and its extensions outperform heuristic schemes by at least 30%.

Due to the difficulty of obtaining a prior knowledge of contents' popularities in real scenarios, designing an online algorithm that does not require such knowledge is desirable. To this end, we presented the Online Collaborative Caching (OCC) algorithm for the Multi-cell coordinated cellular networks in Chapter 4. Collaboration means that a base station can get the content from another base station through the backhaul links instead of getting the content form the origin source. Our algorithm decides which base stations are to cache the contents and from which base station a user's request is satisfied with the objective of minimizing the total cost paid by the content providers. The algorithm works on a per-request basis, where upon the arrival of a request for a content, the Mobility Management Entity (MME) executes the algorithm, updates the value of a potential function of each base station, decides to cache the content if the value of the potential function exceeds the caching cost, and relays its decision to the

base stations. The potential function is a measure of how beneficial it is to cache at a base station. We showed that our algorithm achieves a competitive ratio that is close to the optimal. Our extensive simulations showed that the collaborative caching schemes provide higher savings than the non-collaborative caching scheme, which means that applying the simple online algorithm can lead to lower costs (in terms of the monetary costs paid by the content providers) at very low complexity compared to solving the non-collaborative optimization problem.

Finally, we considered content caching in Content Centric Networks (CCN) in Chapter 5. Routing in CCN is based on the content's name instead of the IP address of the content's source. Therefore, a content can be retrieved from surrounding nodes that are not necessarily on the path to the content's origin server. For the CCN network, we presented Online algorithm for Caching in CCN (OC^3N) that decides the content placement in the caches and which cache to satisfy the request in order to minimize the total cost paid by the content providers. The algorithm works on a per-request basis, does not require a prior knowledge of the contents' popularities, and can be implemented in a distributed way. Upon the arrival of a request for a content at a node, that node becomes responsible of executing the algorithm. Through exchanging messages with other nodes within the vicinity, the node collects the necessary information to execute the algorithm. After a decision is made, the node relays the decision along with necessary information to the participating nodes, which in turn use those information to update their own potential function. For the case where the caching cost changes periodically, and the content items are evicted at the end of the period, our theoretical analysis showed that our algorithm achieves a constant

competitive ratio. For the cases where the caching cost does not change, we proposed a heuristic based on OC^3N where the content items remain in the cache for a fixed time. The conducted simulations showed that our algorithm can achieve up to 65% less cost than widely used caching schemes in CCN such as Leave Copy Down (LCD) and Leave Copy Everywhere (LCE).

6.2 Future Directions

Some of the future possible directions for this research is to investigate the performance of the proposed algorithms under different pricing models, where the economic interaction between the ISP and the content providers can be different depending on the pricing policy adopted by the ISP. Another possible direction is to investigate the effect of having different durations for caching different contents and to design online algorithms with provable performance guarantees to address this case.

Considering content replacement when designing the online algorithms is another possible direction. Although replacement has been considered in this dissertation for some of the algorithms, these were heuristics with no provable performance guarantees. Providing in-depth analysis, considering replacement when designing online algorithms, and investigating the effects of content replacement on the economic interaction between the ISP and the content providers are some of the major challenges.

The online algorithms designed in this dissertations focused on minimizing the total cost. In the case where the algorithms make a decision of not caching a content at a new location, some of the future requests for the same content may have to retrieve the content from farther nodes, which introduces extra delays in content

delivery. Investigating the trade-off between minimizing the total cost and the average delay experienced by end users and designing online algorithms while considering the average delay is another possible future research direction.

Although the algorithms proposed in this dissertations are designed for content caching, these algorithms can be applied to other domains such as cloud resource management in support of Internet of Things and prioritized data processing in Vehicular Ad hoc Networks (VANETs), both of which are currently being investigated. Another application of the proposed algorithms is in the emerging paradigm of Network Function Virtualization (NFV). Our algorithms can be applied in the case where the placement of a single virtual network function (i.e., at which node the virtual function should be instantiated) is to be determined. Handling virtual network function chaining is another possible future research direction.

REFERENCES

[1] C. V. N. Index, "Cisco visual networking index: Global mobile data traffic forecast update, 2014–2019." [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html 2015, (Accessed on March 2015).

[2] A. Vakali and G. Pallis, "Content delivery networks: Status and trends," *IEEE Internet Computing*, vol. 7, no. 6, pp. 68–74, 2003.

[3] A.-M. K. Pathan and R. Buyya, "A taxonomy and survey of content delivery networks," *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, p. 4, 2007.

[4] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[5] J. Rajahalme, M. Särelä, P. Nikander, and S. Tarkoma, "Incentive-compatible caching and peering in data-oriented networks," *ACM CoNEXT Conference*, p. 62, 2008.

[6] T.-M. Pham, S. Fdida, and P. Antoniadis, "Pricing in information-centric network interconnection," *IEEE IFIP Networking Conference*, pp. 1–9, 2013.

[7] Y. Kim and I. Yeom, "Performance analysis of in-network caching for content-centric networking," *Elsevier Computer Networks*, vol. 57, no. 13, pp. 2465–2482, 2013.

[8] D. Dash, V. Kantere, and A. Ailamaki, "An economic model for self-tuned cloud caching," *25th IEEE International Conference on Data Engineering*, pp. 1687–1693, 2009.

[9] V. G. Douros, S. E. Elayoubi, E. Altman, and Y. Hayel, "Caching games between content providers and internet service providers," *Elsevier Perfomance Evaluation (PEVA) Journal*, 2016.

[10] F. Pantisano, M. Bennis, W. Saad, and M. Debbah, "In-network caching and content placement in cooperative small cell networks," *1st IEEE International Conference on 5G for Ubiquitous Connectivity (5GU)*, 2014.

[11] A. Khreishah and J. Chakareski, "Collaborative caching for multicell-coordinated systems," *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 257–262, 2015.

[12] A. Araldo, M. Mangili, F. Martignon, and D. Rossi, "Cost-aware caching: optimizing cache provisioning and object placement in ICN," *IEEE Global Communications Conference*, pp. 1108–1113, 2014.

[13] P. Ostovari, J. Wu, and A. Khreishah, "Efficient online collaborative caching in cellular networks with multiple base stations," *13th IEEE International Conference on Mobile Ad hoc and Sensor Systems*, 2016.

[14] A. Jiang and J. Bruck, "Optimal content placement for en-route web caching," *IEEE Second International Symposium on Network Computing and Applications*, pp. 9–16, 2003.

[15] 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, 2014.

[16] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," *5th ACM international conference on Emerging networking experiments and technologies*, pp. 1–12, 2009.

[17] C. Bernardini, T. Silverston, and O. Festor, "MPC: Popularity-based caching strategy for content centric networks," *IEEE International Conference on Communications (ICC)*, pp. 3619–3623, 2013.

[18] E. J. Rosensweig and J. Kurose, "Breadcrumbs: efficient, best-effort content location in cache networks," *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 2631–2635, 2009.

[19] J. Li, H. Wu, B. Liu, J. Lu, Y. Wang, X. Wang, Y. Zhang, and L. Dong, "Popularity-driven coordinated caching in named data networking," *8th ACM/IEEE symposium on Architectures for Networking and Communications Systems*, pp. 15–26, 2012.

[20] M. D. Ong, M. Chen, T. Taleb, X. Wang, and V. Leung, "FGPC: fine-grained popularity-based caching design for content centric networking," *17th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 295–302, 2014.

[21] J. Li, B. Liu, and H. Wu, "Energy-efficient in-network caching for content-centric networking," *IEEE Communications Letters*, vol. 17, no. 4, pp. 797–800, 2013.

[22] J. Llorca, A. M. Tulino, K. Guan, J. Esteban, M. Varvello, N. Choi, and D. C. Kilper, "Dynamic in-network caching for energy efficient content delivery," *IEEE INFOCOM Proceedings*, pp. 245–249, 2013.

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

[24] N. Zhang, M. Kimmerlin, J. Costa-Requena, and H. Hämmäinen, "Cost efficiency of mobile in-network caching," *International Journal of Network Management*, vol. 26, no. 1, pp. 44–55, 2016.

[25] F. De Pellegrini, A. Massaro, L. Goratti, and R. El-Azouzi, "Competitive caching of contents in 5g edge cloud networks," *arXiv preprint arXiv:1612.01593*, 2016.

[26] A. Araldo, D. Rossi, and F. Martignon, "Cost-aware caching: Caching more (costly items) for less (isps operational expenditures)," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1316–1330, 2016.

[27] Z. Li, Q. Liao, and A. Striegel, "On the economics of mobile content pre-staging." [Online]. Available: http://people.cst.cmich.edu/liao1q/papers/sdp16.pdf (Accessed on December 2016).

[28] M. Hajimirsadeghi, N. B. Mandayam, and A. Reznik, "Joint caching and pricing strategies for popular content in information centric networks," *arXiv preprint arXiv:1609.00852*, 2016.

[29] T. X. Tran, P. Pandey, A. Hajisami, and D. Pompili, "Collaborative multi-bitrate video caching and processing in mobile-edge computing networks," *arXiv preprint arXiv:1612.01436*, 2016.

[30] R. T. Ma and D. Towsley, "Cashing in on caching: On-demand contract design with linear pricing," *CoNext*, 2015.

[31] S. Li, J. Xu, M. van der Schaar, and W. Li, "Trend-aware video caching through online learning," *IEEE Transactions on Multimedia*, vol. 18, no. 12, pp. 2503–2516, 2016.

[32] X. Guan and B.-Y. Choi, "Push or pull? toward optimal content delivery using cloud storage," *Elsevier Journal of Network and Computer Applications*, vol. 40, pp. 234–243, 2014.

[33] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," *ACM ICN workshop on Information-centric networking*, pp. 55–60, 2012.

[34] M. Fiore, F. Mininni, C. Casetti, and C. Chiasserini, "To cache or not to cache?" *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 235–243, 2009.

[35] J. Dai, Z. Hu, B. Li, J. Liu, and B. Li, "Collaborative hierarchical caching with dynamic request routing for massive content distribution," *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 2444–2452, 2012.

[36] N. Laoutaris, G. Zervas, A. Bestavros, and G. Kollios, "The cache inference problem and its application to content and request routing," *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 848–856, 2007.

[37] P. Blasco and D. Gunduz, "Learning-based optimization of cache content in a small cell base station," *arXiv preprint arXiv:1402.3247*, 2014.

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

[39] J. Erman, A. Gerber, M. Hajiaghayi, D. Pei, S. Sen, and O. Spatscheck, "To cache or not to cache: The 3G case," *IEEE Internet Computing*, vol. 15, no. 2, pp. 27–34, 2011.

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

[41] P. Ostovari, A. Khreishah, and J. Wu, "Cache content placement using triangular network coding," *IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1375–1380, 2013.

[42] N. Karamchandani, U. Niesen, M. A. Maddah-Ali, and S. Diggavi, "Hierarchical coded caching," *IEEE International Symposium on Information Theory*, pp. 2142–2146, 2014.

[43] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE International Symposium on Information Theory Proceedings (ISIT)*, pp. 1077–1081, 2013.

[44] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. Leung, "Cache in the air: exploiting content caching and delivery techniques for 5G systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[45] X. Wang, X. Li, V. C. Leung, and P. Nasiopoulos, "A framework of cooperative cell caching for the future mobile networks," *48th IEEE Hawaii International Conference on System Sciences (HICSS)*, pp. 5404–5413, 2015.

[46] Y. Li, H. Xie, Y. Wen, and Z.-L. Zhang, "Coordinating in-network caching in content-centric networks: Model and analysis," *33rd IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 62–72, 2013.

[47] N. Choi, K. Guan, D. C. Kilper, and G. Atkinson, "In-network caching effect on optimal energy consumption in content-centric networking," *IEEE International Conference on Communications (ICC)*, pp. 2889–2894, 2012.

[48] D. Liu and C. Yang, "Energy efficiency of downlink networks with caching at base stations," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 4, pp. 907–922, 2016.

[49] A. Gharaibeh, A. Khreishah, I. Khalil, and J. Wu, "Distributed online en-route caching," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3455–3468, 2016.

[50] T. Bu and D. Towsley, "On distinguishing between internet power law topology generators," *21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 2, pp. 638–647, 2002.

[51] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature, Nature Publishing Group*, vol. 393, no. 6684, pp. 440–442, 1998.

[52] "Mapping the unmappable: Visual representations of the internet as social constructions." [Online]. Available: https://scholarworks.iu.edu/dspace/bitstream/handle/2022/171/wp00-05B.html (Accessed on May 2014).

[53] "Air miles calculator." [Online]. Available: http://www.airmilescalculator.com/ (Accessed on May 2014).

[54] V. Sivaraman, A. Vishwanath, Z. Zhao, and C. Russell, "Profiling per-packet and per-byte energy consumption in the netfpga gigabit router," *IEEE Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 331–336, 2011.

[55] "Soda: Solar radiation data." [Online]. Available: http://www.soda-pro.com/web-services/radiation/helioclim-4 (Accessed on May 2014).

[56] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A provably efficient online collaborative caching algorithm for multicell-coordinated systems," *IEEE Transactions on Mobile Computing*, vol. 15, no. 8, pp. 1863–1876, 2016.

[57] Y. Chen, B. Liu, Y. Chen, A. Li, X. Yang, and J. Bi, "Packetcloud: an open platform for elastic in-network services," *8th ACM International Workshop on Mobility in the Evolving Internet Architecture*, pp. 17–22, 2013.

[58] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[59] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, "Impact of cloudlets on interactive mobile cloud applications," *16th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pp. 123–132, 2012.

[60] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[61] Wikipedia, "Customer attrition." [Online]. Available: http://en.wikipedia.org/wiki/Customer_attrition (Accessed on January 2015).

[62] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access (E-UTRAN); Overall description; Stage 2." [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/36300.htm 2008, (Accessed on September 2014).

[63] L. E. Li, Z. M. Mao, and J. Rexford, "Toward software-defined cellular networks," *IEEE European Workshop on Software Defined Networking (EWSDN)*, pp. 7–12, 2012.

[64] Y. Bartal, "Probabilistic approximation of metric spaces and its algorithmic applications," *37th IEEE Annual Symposium on Foundations of Computer Science*, pp. 184–193, 1996.

[65] IBM, "Cplex optimizer." [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/ 2015, (Accessed on January 2015).

[66] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," *18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 1, pp. 126–134, 1999.

[67] A. Gharaibeh, A. Khreishah, and I. Khalil, "An $\mathcal{O}(1)$-competitive online caching algorithm for content centric networking," *IEEE International Conference on Computer Communications (INFOCOM)*, 2016.

[68] A.-H. Mohsenian-Rad and A. Leon-Garcia, "Energy-information transmission tradeoff in green cloud computing," *Citeseer Carbon*, vol. 100, p. 200, 2010.

[69] Amazon, "Amazon ec2 pricing." [Online]. Available: http://aws.amazon.com/ec2/pricing/ (Accessed on June 2015).

[70] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta algorithms for hierarchical web caches," *IEEE International Conference on Performance, Computing, and Communications*, pp. 445–452, 2004.