Design, Modeling, and Evaluation of Efficient Caching Mechanisms for Content Dissemination Networks

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2015

Satoshi IMAI

List of publications

Journal Papers

1. Satoshi Imai, Kenji Leibnitz, Masayuki Murata, “Energy Efficient Data Caching for Content Dissemination Networks”, Journal of High Speed Networks, DOI:10.3233/JHS-130474, vol. 19, no. 3, pp. 215-235, October, 2013.

2. Satoshi Imai, Kenji Leibnitz, Masayuki Murata, “Adaptive TTL Control to Minimize Resource Cost in Hierarchical Caching Networks”, to appear in IEICE Transactions on Information and Systems, March, 2015.

Refereed Conference Papers

1. Satoshi Imai, Kenji Leibnitz, Masayuki Murata, “Energy Efficient Content Locations for In-Network Caching”, in *Proceedings of* 18th Asia-Pacific Conference on Communications (APCC 2012), Jeju, Korea, Oct. 2012.

2. Satoshi Imai, Kenji Leibnitz, Masayuki Murata, “Energy-Aware Cache Management for Content-Centric Networking”, in *Proceedings of* First International WorkShop on Energy-Aware Systems, Communications and Security, Barcelona, Spain, March 2013.

3. Satoshi Imai, Kenji Leibnitz, Masayuki Murata, “Modeling of Content Dissemination Networks on Multiplexed Caching Hierarchies”, in *Proceedings of* the 13th International Conference on Networks, Nice, France, 23-27 Feb. 2014.

Non-Refereed Technical Papers

1. Satoshi Imai, Kenji Leibnitz, Masayuki Murata, “Energy Awareness based on Cache Allocation Management in Content-Centric Networking”, *Technical Report of IEICE (IN2011-140)*, vol. 111, no. 469, pp. 19–24, March 2012 (in Japanese).

Preface

As diversity of services and applications on networks is increasing, broadband traffic is growing as well. To reduce the network traffic and efficiently use the resources of network systems, edge computing and *Content Delivery Networks* (CDN) are well known as technology for facilitating the processing of application programs/contents at edge servers near users in the network. Furthermore, In-network caching technologies like *Content-Centric Networking* (CCN) for *Information-Centric Networking* (ICN) are expected to reduce the network traffic and improve the service quality such as communication latency by storing content data on network nodes near to users.

In these architectures for content dissemination networks, each node has caching functionality and can achieve an adaptive content delivery. Therefore, the network traffic and communication quality are influenced by the cache locations because content is generated by many publishers at various locations and causes different amount of traffic depending on its popularity. Moreover, the caching performance depends on the memory size of each node, delivery routes, and request distribution of content. On the other hand, traditional caching mechanisms can be managed by many methods, like *Least Recently Used* (LRU), *Least Frequently Used* (LFU) policies, but also by more advanced methods for specific cache types that deal well with *one-timers*, which are accessed only once and are not hit while in the cache, or by the limited lifetime of content in the cache called *Time-To-Live* (TTL).

Therefore, to realize effective caching networks, it becomes important to grasp the cache characteristics of each node and to manage system resources, taking into account energy and usage efficiency for storage and network bandwidth, which are influenced by the characteristics of each cache

node. This thesis discusses methodologies for improving the performance in design and cache management for content dissemination networks. As individual sub-problems, we investigate the issues for energy efficiency, analysis of efficient caching mechanisms for *one-timers* and the TTL-based caching mechanism, and controllability of system resources in content dissemination networks.

Regarding the first issue for improving energy efficiency, we propose a design method which derives the most energy efficient cache locations of content and a distributed cache mechanism like CCN to search for energy efficient cache locations to be close to the optimal solution. Simulation results show that the distributed cache mechanism is able to reduce the power consumption of the target network by at most 26% of traditional LFU policy and to achieve similar energy efficiency within an increment less than 8% of the optimal caching.

To solve the next issue of analyzing cache characteristics, we focus on the 2Q and *Adaptive Replacement Caching* (ARC) mechanisms as effective caching for *one-timers* and propose approximation models, which can statistically analyze the influences of memory size and request distribution on the cache performance. As a result of evaluating the approximation accuracy of the proposed models, we show that our proposed method is able to estimate the hit probability with an error of less than 2% for simulation results at a significantly shorter time.

We further research into the issue for analyzing cache aging techniques based on *Time-To-Live* (TTL) of content in hierarchical caching. The TTL-based caching facilitates analyzing cache characteristics and can realize appropriate resource management by efficiently setting TTLs. We propose a theoretical model, which can analyze the impact of TTL-based caching on network resources and cache performance in the distributed cache systems, with high accuracy of less than 3% error of the simulation results. Furthermore, we introduce a caching mechanism using energy efficient TTLs and show its effectiveness by the model-based analysis.

We finally investigate the issue of the controllability of system resources in caching networks. As an enhanced mechanism of distributed TTL-based caching, we propose an adaptive control mechanism of the TTL value of content by using predictive models which can estimate the impact of the TTL values on the resource cost of a caching network. Furthermore, we show the effectiveness of the proposed mechanism. Through this research, we have confirmed that the proposed predictive models can obtain good approximations of the control characteristics by the TTL values and that

the proposed control mechanism can efficiently manage system resources in caching systems by always searching for the minimum cost in the given range of TTLs.

Acknowledgments

This thesis could not have been accomplished without the assistance of many people, and I would like to acknowledge all of them.

First of all, I would like to express my great gratitude to my supervisor, Professor Masayuki Murata, for his generous guidance, insightful comments, and meaningful discussion. I was able to complete my thesis owing to his kind guidance, timely encouragement, and valuable advice.

Also, I would especially like to express my sincere appreciation to Dr. Kenji Leibnitz of the Center for Information and Neural Networks (CiNet), National Institute of Information and Communications Technology, and Osaka University. He directed me to the appropriate perspective and inspired me to aim at higher goals. My study would not have been possible without his continuous care and support. I am deeply grateful to him.

I am heartily thankful to the members of my thesis committee, Professor Takashi Watanabe, Professor Toru Hasegawa, Professor Teruo Higashino, and Professor Morito Matsuoka of the Graduate School of Information Science and Technology, Osaka University, for their multilateral reviews and perceptive comments.

Furthermore, I would like to acknowledge Mr. Tsuguo Kato, Mr. Hideyuki Miyata, Mr. Toru Katagiri, Mr. Akira Chugo, Mr. Toshio Soumiya, and Ms. Akiko Yamada of Network Systems Laboratories, Fujitsu Laboratories Ltd., for their support and encouragement during the course of this study.

Furthermore, I appreciate all the members of the Network Systems Engineering Laboratory, Fujitsu Laboratories Ltd., for their support and friendship. I learned a lot through a variety of discussions and interactions with them.

Finally, I deeply thank my parents, my wife Chieko, my daughter Ayano, and my son Hiroto, for their understanding and hearty support and encouragement in my daily life. This work would not have been achieved without them.

Contents

List of publications **i**

Preface **iii**

Acknowledgments **vii**

1 Introduction **1**

1.1 Background 1

1.2 Outline of Thesis 5

2 Design, Modeling, and Evaluation of Energy Efficient Cache Management **9**

2.1 Issues and Approaches for Energy Efficient Cache Management 9

2.2 Optimal Cache Locations 11

2.2.1 Design Flow 11

2.2.2 Optimization Model 11

2.3 Distributed Cache Mechanism 15

2.3.1 Process Flow 15

2.3.2 Distributed Algorithm for Threshold Design 17

2.4 Evaluation 19

2.4.1 Tradeoff of Power Consumption 21

2.4.2 Effectiveness of the Distributed Cache Mechanism 24

2.4.3 Power Consumption and Memory Usage 25

2.4.4 Cache Performance 29
2.5 Summary 33

3 Modeling and Evaluation of Node-Level Cache Management 35
3.1 Issue and Approach for Analysis of Node-Level Cache Management 35
3.2 Caching Mechanisms for One-Timers 37
3.2.1 2Q 38
3.2.2 Adaptive Replacement Caching (ARC) 39
3.3 Proposed Approximation Models 41
3.3.1 Approximation for Simplified 2Q 41
3.3.2 Approximation for Full 2Q 44
3.3.3 Approximation for ARC 47
3.4 Evaluation 50
3.4.1 Approximation Accuracy of Cache Hit Probability in 2Q 51
3.4.2 Approximation Accuracy of Cache Hit Probability in FRC and ARC 52
3.4.3 Comparison of Performance in Each Caching Mechanism 55
3.5 Summary 56

4 Modeling and Evaluation of Static TTL Management in Hierarchical Caching 59
4.1 Issue and Approach for Analysis of Static TTL Management 59
4.2 Analytical Model 60
4.2.1 Memory Usage 63
4.2.2 Transmission Data 64
4.2.3 Power Consumption 64
4.2.4 Cache Hit Ratio 65
4.2.5 Average Hop Length 65
4.3 Evaluation using the Proposed Model 65
4.3.1 Verification of the Proposed Model 67
4.3.2 Impact of TTL 68

4.4 Application to a Caching Mechanism using Energy Efficient TTLs 70
4.5 Summary 75

5 Design, Modeling, and Evaluation of Adaptive TTL Management in Hierarchical Caching **77**

5.1 Issue and Approach for Adaptive TTL Management 77
5.2 System Model 80
5.2.1 Request Propagation Model 81
5.2.2 Resource Cost Model 82
5.3 Proposed TTL Controller 83
5.3.1 Design Algorithm of TTL Controller 83
5.3.2 Distributed Prediction 88
5.4 Evaluation 93
5.4.1 Comparison of the Proposed Model and Simulation 95
5.4.2 Effectiveness of the Proposed Mechanism 96
5.4.3 Verification of Optimality 99
5.4.4 Impact of the TTL Control on Cache Performance 101
5.4.5 Adaptability of TTL Control 102
5.5 Summary 103

6 Conclusion **105**

Bibliography **109**

List of Figures

1.1 Caching hierarchies for each origin site 3

2.1 Example of definition of route candidates 12
2.2 Network model 12
2.3 Constraint conditions 15
2.4 Example of creating a caching hierarchy and the sub-tree structure 16
2.5 Bottom-up process 17
2.6 Test topologies 20
2.7 Request distribution of the content $R_{k,*}$ 20
2.8 Origin sites of content 21
2.9 Power consumption for a chunk of each content ID and the ratio of power consumption to that of optimal caching (Zipf:$\alpha = 1.2$, TopologyA) 22
2.10 Power consumption for a chunk of content ID:398 (Zipf:$\alpha = 1.2$, TopologyA) . . . 22
2.11 Power consumption for a chunk of each content ID and the ratio of power consumption to that of optimal caching (Zipf:$\alpha = 1.2$, TopologyB) 23
2.12 Power consumption for a chunk of content ID:395 (Zipf:$\alpha = 1.2$, TopologyB) . . . 23
2.13 Example of time change of total power consumption for content following a Zipf distribution with $\alpha = 1.2$ (solid line: total power consumption / dashed line: cache allocation power in the total power consumption) 25
2.14 Average power consumption when the memory size is changed 27

2.15 The cumulative power consumption of chunks of content for threshold-based caching and optimal caching when the memory size is infinite 28
2.16 The total power consumption when the memory size is infinite and the request distribution is changed in Topology B 29
2.17 Request distribution of the content (dasched line: $r = 150$, solid line: $r = 100$, dotted line: $r = 50$) 29
2.18 Average used memory when the memory size is changed 30
2.19 Cache hit ratio in the network when the memory size is changed 31
2.20 Average hop length when the memory size is changed 32

3.1 Queue structure of Simplified 2Q 38
3.2 Queue structure of Full 2Q 38
3.3 Queue structure of ARC 40
3.4 Queue model of Simplified 2Q 41
3.5 Queue model of Full 2Q 44
3.6 Queue model of ARC 47
3.7 Content popularity 50
3.8 Comparison of cache hit probability per object in Simplified 2Q estimated by the proposed model and measured by simulations 51
3.9 Cache hit probability per object in Full 2Q estimated by the proposed model and measured by simulations 52
3.10 Cache hit probability per object in FRC(1) estimated by the proposed model and measured by simulations 53
3.11 Average hit probability of all objects estimated by the FRC model when changing p and measured by simulations of ARC 54
3.12 Box plot of values of p in simulations of ARC ($C = 1000$ and 2000) 54
3.13 Cache hit probability per object estimated by the proposed model of FRC(1) and measured by simulations of ARC 55
3.14 Cache hit probability of all objects estimated by each approximation model 56

4.1 Traditional TTL-based caching 61
4.2 Cache hit ratio when the requests per content with the rate λ input to a *CN* at an exponentially distributed interval and set "total request rates of a content item [requests/sec]" and "TTL [sec]" to various patterns 61
4.3 An example of matrices $\mathbf{\Lambda}^c$, $\mathbf{R}^c$, and D^c for the request propagation on the delivery tree having origin 1 62
4.4 Evaluation conditions r_j^c 67
4.5 Cache performance estimated by the proposed model and calculated by simulations for different content ids 68
4.6 Box plot of memory usage at each *CN* when the TTL value is changed 69
4.7 Power consumption of the network when the TTL value is changed 70
4.8 Cache performance when the TTL value is changed 71
4.9 Comparison of cache hit ratio at a *CN* with threshold-based caching and TTL-based caching 72
4.10 Energy efficient TTL for each topology 73
4.11 Power consumption for energy efficient TTL-based caching and threshold-based caching 73
4.12 Cache performance for energy efficient TTL-based caching and threshold-based caching 74
4.13 Box plot of memory usage for TTL-based caching and threshold-based caching . . 75

5.1 TTL-based caching 79
5.2 The tradeoff between *storage cost* and *bandwidth cost* 79
5.3 An example of matrices $\mathbf{\Lambda}^c$, $\mathbf{R}^c$, and D^c for the request propagation on the delivery tree having origin 1 82
5.4 Outline of the proposed control mechanism 84
5.5 Distributed prediction-based control mechanism 88
5.6 An example of the distributed prediction 90
5.7 Tree topology for evaluation 94

5.8 Request distribution from each site 94
5.9 Total resource cost estimated by the proposed model and measured by simulations for different object ids in $C_s : C_b = 0.1 : 0.9$ 96
5.10 Convergence values of resource cost and TTLs for content objects 97
5.11 Cumulative resource cost of all objects and the effectiveness of reducing the total resource cost (initial cost, controlled cost, cost when setting TTLs to 60 and 120 sec) 98
5.12 Tradeoff between storage cost and bandwidth cost for object ids: {200, 250, 300} with $\alpha = 0.8$ in $C_s : C_b = 0.1 : 0.9$ 99
5.13 Control trajectory and time change of TTLs for objects with $\alpha = 0.8$ in $C_s : C_b = 0.1 : 0.9$ and with $\alpha = 1.2$ in $C_s : C_b = 0.03 : 0.97$ when the initial TTL is set to 1 sec 100
5.14 Control trajectory for object with $\alpha = 0.8$ in $C_s : C_b = 0.1 : 0.9$ and with $\alpha = 1.2$ in $C_s : C_b = 0.03 : 0.97$ when the initial TTL is set to 150 sec 101
5.15 Cache miss ratio and average hop length of all objects when converging at the minimum cost 102
5.16 Control trajectory for object ids: {200, 250, 300} in $C_s : C_b = 0.1 : 0.9$ 103

List of Tables

2.1 Variables in the optimal design of cache locations 13
2.2 Variables in the distributed cache mechanism 19
2.3 Power density parameters. 19

4.1 Variables in the proposed model 66

5.1 Variables used in the proposed model 85
5.2 Power cost parameters 93
5.3 Comparison between numerical evaluation and simulation 95

Chapter 1

Introduction

1.1 Background

The growing number of Information and Communication Technology (ICT) services on networks is currently leading to a huge increase in network traffic. To reduce network traffic and efficiently use ICT resources, edge computing [1, 2, 3] is well known as technology for facilitating the processing of application programs/contents at edge servers near users in the network. As a result, edge computing can also distribute the computation load and databases to each edge server. An edge server has the functionality of generating user-specific contents and the origin server manages the distributed system and master data for edge computing.

As one of edge computing technologies for reducing network traffic, *Content Delivery Network* (CDN) architectures [4, 5] in metro/access networks, such as Akamai-CDN [6] or web proxies, are well known. The CDNs can manage the content delivery at the edge of the networks by allocating content replicas in cache servers, which are in geographical proximity to users, and it is expected that CDNs not only reduce the network traffic but also improve communication quality such as latency and throughput, when delivering content.

Furthermore, various caching architectures [7, 8] for *Information-Centric Networking* (ICN), in which each node has caching functionality for content, have been actively discussed. In *Content-Centric Networking* (CCN) [9] for ICN, content publishers advertise newly released content from

the origin site along predefined routes. CCN is a receiver-driven protocol where *Data* is only sent in response to a user's request. Each content is divided into chunks of fixed size and these chunks are cached on content routers along the content delivery route.

The content dissemination mechanism in CCN has the following features.

1. **Content Advertisement:** Content publishers advertise newly released content from the origin site of the content along predefined routes.

2. **Content Discovery/Delivery:** A content request (*Interest*) is forwarded on each content router based on the *Forwarding Information Base* (FIB) until the requested content can be found. An *Interest* forwarded on a router is added to the *Pending Interest Table* (PIT) in order to remember the interface on which to send back the replies (*Data*). When the requested content is found on a router, *Data* of the content are transmitted based on the PIT.

3. **Content Storage:** *Data* are cached on all routers along the transmission route based on the specific replacement policy such as Least Recently Used (LRU) or Least Frequently Used (LFU).

Caching technologies like CCN for ICN can reduce network traffic by storing content data in many locations. The distributed mechanisms like CCN can automate the placement and delivery of content data for efficient content delivery. Meanwhile, network traffic is influenced by the cache locations because each content generates a different amount of traffic depending on its popularity. Additionally, many storages such as DRAM or SSD are required for content caching. Most caching networks have hierarchical tree structures, logically or physically connected by cache nodes (*CN*s) [10, 11, 12], and each content is delivered on the caching hierarchy rooted at each origin site of content (cf. Fig. 1.1). Likewise in CCN, each cache node autonomously constructs a caching hierarchy rooted at an origin site. In these distributed mechanisms, the caching hierarchy is constructed by routes between the origin site, caching *CN*s, and users, such that less popular content is cached on *CN*s near to the origin site and more popular content is cached on *CN*s near to users.

On the other hand, caching mechanisms can be managed by traditional methods, like LRU,

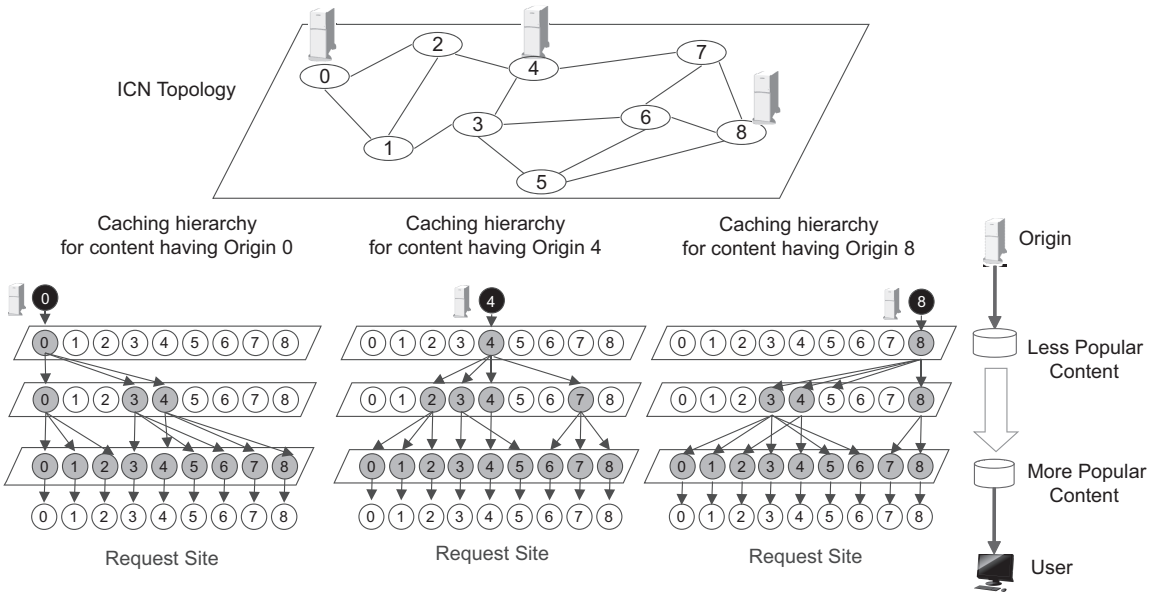


Figure 1.1: Caching hierarchies for each origin site

LFU, and *First In, First Out* (FIFO) replacement policies. Additionally, the general request distribution in content dissemination networks is heavy-tailed containing many objects called *one-timers*, which are accessed only once and are not hit while in the cache. However, in the traditional caching mechanisms, the *one-timers* may deteriorate the whole cache performance by causing inefficient cache replacement which discards more popular content. Beside the problem of *one-timers*, the content, such as video streaming, generally has a limited lifetime. Therefore, caching mechanisms can use a limited period of time, called *Time-To-Live* (TTL), for content. Moreover, TTL-based caching can improve the adaptability of cache management per object compared with the existing caching policies.

Taking into account the aforementioned background, to establish strategies on improving the performance in system design and cache management for content dissemination networks, we focus on solving the following issues.

Energy Efficiency of Cache Management

In order to realize energy efficient resource management in content dissemination networks, it is important to manage the appropriate cache locations in consideration of the balance issue

of traffic volume for delivering each content and memory used for storing content replicas by locally collaborating with each node in accordance with the distributed mechanism like CCN for ICN.

Analysis of Cache Management for *One-Timers*

To avoid the influence of cache pollution due to *one-timers*, effective caching mechanisms having separate queues for *one-timers*, such as 2Q [13] and *Adaptive Replacement Caching* (ARC) [14, 15], have been proposed. However, in previous work, caching mechanisms like 2Q and ARC are only discussed with respect to the general cache performance for various workloads as a result of the implementation of the mechanisms. Therefore, it is required to reveal the characteristics in the caching mechanisms like 2Q and ARC and analyze the influence of cache pollution due to *one-timers*.

Analysis of TTL-based Cache Management

TTL-based caching is useful for efficient cache management per content object. However, the characteristics in the distributed cache mechanism by using TTL-based caching depend on the multiplexed caching hierarchies shown in Fig. 1.1. Therefore, it is also important to grasp the impact of TTL-based caching on network resources and performance.

Controllability of TTL-based Cache Management

On the assumption that ICN services are provided on the network virtualization platform, the caching services can manage resource cost like monetary cost for each content/application by providing the necessary system resources, such as storage and communication bandwidth, in the caching network. TTL values are useful for efficiently managing system resources in distributed cache systems. However, to realize the efficient resource control by using TTL, it becomes an issue to decide how much of the TTL value of content should be increased/decreased to reduce the resource cost of the caching network.

Therefore, this thesis focuses on these issues for improving energy efficiency, analysis of the characteristics of caching mechanisms for *one-timers* and TTL-based caching, and controllability of system resources by using the TTL value, in content dissemination networks.

1.2 Outline of Thesis

This thesis investigates the issues of system design for energy efficient cache management, modeling of cache characteristics and distributed cache management at node level, and control methodology of system resources in content dissemination networks including various mechanisms for Information-Centric Networks including edge computing, content delivery networks, and CCN.

Design, Modeling, and Evaluation of Energy Efficient Cache Management [16, 17, 18]

The growth in network traffic also increases the power consumption of network systems year by year. In the reports of the Ministry of Economy, Trade, and Industry [19], broadband traffic in Japan is growing at an annual rate of 25% and network power consumption is expected to occupy 20% of the total ICT power consumption by 2025.

Recently, power-saving mechanisms of network devices [20] have been studied in order to realize *Energy Proportional Networks* [21, 22] in which power consumption of each device is proportional to its usage. In addition, energy efficiency can be improved by reducing the traffic flow in the network, because the traffic decrease can improve the effect of the power-saving mechanism in each device or can prevent that frequent incremental deployments of the network devices are required.

Therefore, in order to realize energy efficiency of the network under the condition that network devices and memory can be deployed in proportion to their usage, the appropriate cache allocation should be executed in consideration of the *cache allocation power*, i.e., the memory power consumed by storing the content and the *traffic transmission power*, i.e., the total power consumed by network devices when data are transmitted.

In Chapter 2, our first topic aims to realize energy efficient cache locations in CCN and we propose a design method to minimize the sum of power consumption of *storage devices for content caching* and power consumption of *network devices for content delivery* [16]. Furthermore, we propose a distributed cache mechanism to locally search for energy efficient cache locations attempting to be close to the optimal cache locations [17]. In the distributed cache mechanism, each node autonomously pre-designs a threshold of request rates of content before cache operation and

judges whether to cache content data by comparing content request rates with the designed threshold. Additionally, we evaluate the energy efficiency of the proposed methods for some scenarios.

Modeling and Evaluation of Node-Level Cache Management

In caching systems for ICN, the communication quality is influenced by the cache performance of each node. Moreover, the cache performance depends on the memory size allocated on each node and content popularity according to the request distribution. Therefore, it is important to analyze the cache characteristics of each node.

As representative caching mechanisms, LRU, LFU, and FIFO policies are well-known and statistical approximations for these mechanisms have been proposed [11, 23]. These approximation models enable us to analyze cache characteristics at high accuracy and faster than computer simulations. Meanwhile in ICN, the content popularity often follows a Zipf-like distribution including many objects called *one-timers* [24, 25, 26] which are accessed only once and are not found in the cache (no hit) when requested. Moreover in single queue management like LRU, LFU, and FIFO, the *one-timers* can cause inefficient cache replacement by discarding more popular content in favor of the *one-timers*. As a result, *one-timers* can deteriorate the whole cache performance. Actually, *one-timers* account for a high percentage in the realistic workloads of web proxies and VoD services and several studies [24, 26, 27, 28, 29] address the problem of *one-timers*. To solve this problem and improve cache performance, effective caching mechanisms having separate queues for *one-timers* such as 2Q and *Adaptive Replacement Caching* (ARC) have been proposed. Moreover, to design efficient caching systems in which each node has limited memory size, it is important to analyze the interaction of these separate queues, which influence each other, in 2Q and ARC.

In Chapter 3, we focus on analyzing the cache characteristics at a node with 2Q and ARC and on approximating the cache characteristics of these mechanisms which have separate queues for *one-timers*. Furthermore, we model statistical characteristics of 2Q and ARC and demonstrate the approximation accuracy of the proposed models by comparing cache performance estimated by the approximation model with that measured by simulations.

Modeling and Evaluation of Static TTL Management in Hierarchical Caching [30]

The persistent storage of content in the limited memory of each node is inefficient because the content, such as video streaming, generally has a limited lifetime. Therefore, caching mechanisms often use a limited period of time, called *Time-To-Live* (TTL) for content. Moreover, TTL-based caching can improve the adaptability of cache management per object compared with the existing policies such as LRU, FIFO, 2Q, or ARC with cache coordination which sorts content by popularity and selects which content should be discarded. However, it is difficult to evaluate the impact of TTL-based caching on network resources and performance because the characteristics in the distributed cache mechanism depend on the caching hierarchies connected by distributed cache nodes. Therefore, we focus on analyzing the impact of the distributed cache mechanism based on TTL-based caching on network resources, such as memory or network devices, and cache performance, such as cache hit ratio and hop length, while delivering content.

In Chapter 4, we propose an analytical model to evaluate the impact of TTL on cache performance and network resources of content in a distributed cache mechanism like CCN. Furthermore, we demonstrate evaluation results for some network scenarios using the proposed model. In addition, we introduce a caching mechanism using energy efficient TTL as one possible application of TTL-based caching and show the effectiveness of the proposed caching mechanism using our model.

Design, Modeling, and Evaluation of Adaptive TTL Management in Hierarchical Caching [31]

Recently, network virtualization technologies such as *Network Function Virtualization* (NFV) have been attracting attention [32, 33]. On the assumption that ICN services are provided on the network virtualization platform, the caching services can manage the resource cost like monetary cost for each content/application by providing necessary system resources, such as storage and communication bandwidth, in the caching network.

In the distributed cache mechanisms, the network traffic and cache characteristics are influenced by the cache locations and depend on the memory size in each cache node on the delivery tree.

Therefore, it is difficult to manage the relationship among resource cost, network resources, such as memory and network devices, and cache characteristics, such as cache hit ratio. As a result, in ICN services, it becomes an important issue to manage the total cost of system resources, such as storage and network bandwidth, which are influenced by cache characteristics of each cache node. Meanwhile, TTL-based caching doesn't only facilitate analyzing the cache characteristics but also provides flexibility and adaptability of cache management per content [11, 34, 35].

In Chapter 5, we first introduce predictive models which estimate the impact of TTL on cache characteristics, network resources, and the resource cost in the distributed cache system. Furthermore, we propose an adaptive TTL control mechanism based on the predictive models to reduce the total resource cost and highlight the control methodology for general hierarchical TTL-based caching as an application of Newton's method [36, 37] for finding the optimum of a nonlinear function.

Finally, Chapter 6 concludes this thesis with directions for future work.

Chapter 2

Design, Modeling, and Evaluation of Energy Efficient Cache Management

2.1 Issues and Approaches for Energy Efficient Cache Management

Reducing network traffic is important because power consumption in networks has been increasing year by year. Meanwhile, in-network caching for ICN is expected to achieve an adaptive content delivery in the network and to reduce traffic by storing content data on each cache node. Therefore, in-network caching is beneficial in view of energy efficiency. However, in order to improve energy efficiency of the network, it is necessary to cache content appropriately in consideration of both power consumption of content caching and transmission of traffic.

Recently, energy efficiency in CCN has been attracting a lot of attention [38, 39, 40]. Lee *et al.* [38, 39] survey the energy efficiency of various network devices deployed in access/metro/core networks. Furthermore, they evaluate the power-saving effect in the entire network when the deployment ratio of CCN-enabled edge/core routers is changed. As a result, they show that CCN is able to improve the energy efficiency of current CDNs.

Furthermore, Guan *et al.* [40] build energy models of traffic transmission power and caching power for content delivery architectures such as "Conventional and decentralized server-based

CDN", "Centralized server-based CDN using dynamic optical bypass", and CCN. Using their energy models, they analyze the energy efficiency of each of these architectures. Those energy models are approximations based on the relations between the topological structure and the average hop-length from all sites to the nearest cache location. The appropriate number of cache locations for content can be estimated in order to save the network power consumption when the target topology, the required average hop-length, and the number of requests for content are given. In those papers, energy efficiency is represented differently depending on the cache locations for content.

However, these works don't discuss the design or search for energy efficient content locations of caching hierarchies, which have different origin sites for content and asymmetric routes. In order to realize energy efficient locations of content which can reduce the sum of *cache allocation power* and *traffic transmission power*, we should consider constraints of multiplexed caching hierarchies of content (cf. Fig. 1.1). Furthermore in CCN, a request should be autonomously executed by some cache nodes (*CN*s) without needing to know the content locations.

In this chapter, we first aim to provide reference locations to realize energy efficiency for cache strategies and introduce a new design method which can derive energy efficient locations of content on constraints of the caching hierarchy rooted at the content's origin site so as to minimize the sum of *cache allocation power* and *traffic transmission power*. Secondly, we show a distributed cache mechanism to search for energy efficient cache locations of content based on the caching hierarchies attempting to be close to the optimal solution. Furthermore, we summarize the details of the proposed methods and combine both approaches into the same framework which makes it easy to compare both. Additionally we show new evaluation results.

The remainder of this chapter is organized as follows. We next describe the design method to solve the cache location problem in Section 2.2 and the distributed cache mechanism to search for energy efficient cache locations in Section 2.3. Section 2.4 demonstrates simulation results and we summarize this chapter in Section 2.5.

2.2 Optimal Cache Locations

In order to provide reference locations to evaluate energy efficiency for cache strategies, we show a 0-1 Integer Linear Programming Problem (0-1 ILP) model [41] to design energy efficient cache locations of chunks of a target content satisfying the constraints of multiplexing caching hierarchies which have a different origin site for content [16]. The model designs cache locations of chunks of content in order of descending popularity to minimize total power consumption based on *Energy Proportional Networks*.

2.2.1 Design Flow

The cache location design for each content is executed in the following steps.

step 1 Get content popularity.

step 2 Select a target content (k-th popular content) in the order of content popularity.

step 3 Extract route candidates to deliver the target content from a cache location to a requesting user on the shortest-path tree rooted at the origin site of the target content and define the route candidates as the design variables, cf. Figs. 2.1 and 2.2.

step 4 Design the optimal cache locations of the target content based on the route candidates in consideration of the pre-designed routes for more popular content which have the same origin site as that of the target content.

step 5 Return to **step 2** and execute the design of the next popular content ($[k+1]$-th popular content).

2.2.2 Optimization Model

The objective function is defined as follows. For a network composed of content routers (cache nodes) and Wavelength Division Multiplexing (WDM) nodes in Fig. 2.2, this model designs cache locations and data (chunks) transmission routes for the delivery tree, which consists of a set of

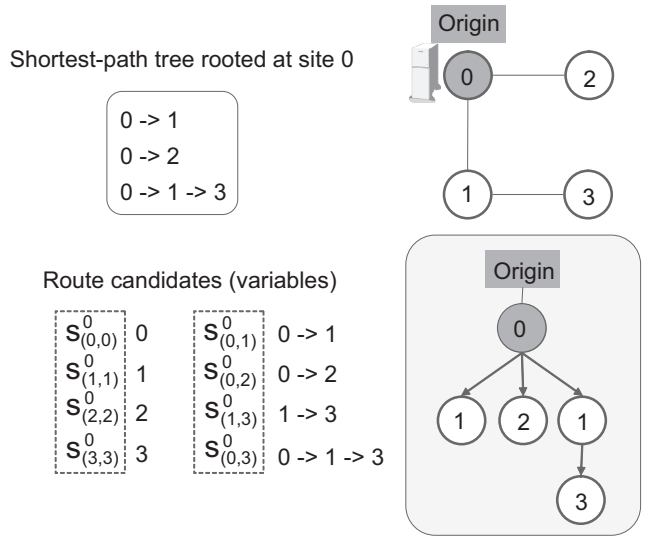


Figure 2.1: Example of definition of route candidates

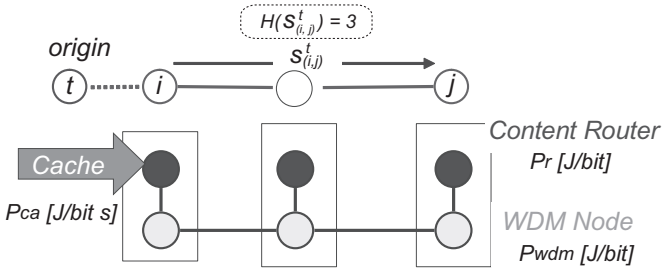


Figure 2.2: Network model

vertices (sites) V and route candidates R_c. The objective is to minimize the sum of cache allocation power Ca_i, i.e., the power for storing a chunk of the target content in *CN* on site i, and *traffic transmission power* $Tr_{k,(i,j)}$, i.e., the total power of content routers and WDM nodes when a chunk of content k is delivered on the route from site i to site j.

$$\textbf{Minimize} \sum_{i \in \mathrm{V}} \{Ca_i \cdot u_i\} + \sum_{(i,j) \in \mathrm{R_c}} \left\{ Tr_{k,(i,j)} \cdot s^t_{(i,j)} \right\} \tag{2.1}$$

The design variables are defined as follows.

- $u_i \in \{0,1\}$ indicates whether or not to store a chunk of the target content on CN_i.

Table 2.1: Variables in the optimal design of cache locations

Variable	Type of Variable	Definition
$R_{k,j}$	Given	The request rate for target content k from a destination site j
P_{ca}	Given	Power density for storage (memory) [J/(bit·s)]
P_r	Given	Power density of a content router [J/bit]
P_{wdm}	Given	Power density of a WDM node [J/bit]
D	Given	Data size of a chunk [bits]
u_i	Design	Binary variable for whether to store a chunk of target content k in CN_i or not
$s^t_{(i,j)}$	Design	Binary variable for whether to select the route from CN_i to CN_j defined on a shortest-path tree rooted at origin site t of a chunk of target content k or not

- $s^t_{(i,j)} \in \{0,1\}$ describes whether or not to select the route from site i to site j defined on the shortest-path tree rooted at the origin site t of a chunk of the (k-th popular) target content.

All variables in the optimization model are summarized in Table 2.1.

Here, the *cache allocation power* Ca_i [J] in 1 sec when a chunk of the target content is cached in CN_i ($u_i = 1$) is defined as

$$Ca_i = 1 \cdot D \cdot P_{ca}, \tag{2.2}$$

where P_{ca} is the memory power density [J/(bit·s)] and D is the chunk size [bits] of the target content.

Furthermore, the *traffic transmission power* $Tr_{k,(i,j)}$ [J] when a chunk of (k-th popular) target content is delivered on the candidate route $s^t_{(i,j)}$ from CN_i to CN_j on the shortest-path tree rooted at origin site t, is defined as

$$Tr_{k,(i,j)} = D \cdot R_{k,j} \cdot (P_r + P_{wdm}) \cdot \mathrm{H}(s^t_{(i,j)}) \tag{2.3}$$

where P_r and P_{wdm} are the power densities [J/bit] of a CN and of a WDM node, respectively, and $R_{k,j}$ is the request rate from site j for the target content. Furthermore, we define $\mathrm{H}(s^t_{(i,j)})$ as the hop-length of route $s^t_{(i,j)}$.

We next define the constraints for the proposed 0-1 ILP model. The transmission routes to site j, which requests the target content having origin root t, should be created.

$$\sum_{i \in \mathrm{V}} s^t_{(i,j)} = 1 \quad \forall j \in \mathrm{V} \tag{2.4}$$

Moreover, the starting site of the transmission route should be the cache location of the target content.

$$s^t_{(i,j)} \leq u_i \quad \forall i \in \mathrm{V},\ \forall (i,j) \in \mathrm{R_c} \tag{2.5}$$

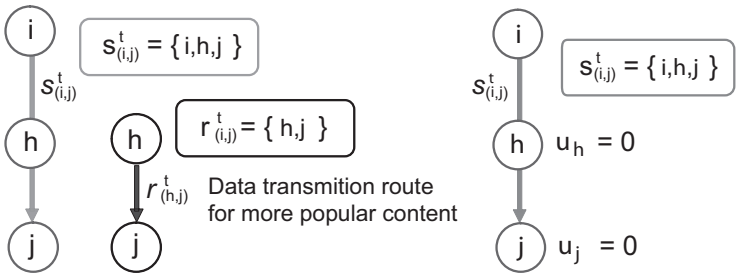
For the hierarchical route constraint, the target content should be cached on the shortest-path tree rooted at its origin site t. Furthermore, there should be more popular content, having the same origin site t as the target content, on the designed transmission route and caching hierarchy should be constructed as shown in Fig. 1.1.

$$s^t_{(i,j)} = 0 \quad \text{if } \mathbf{r}^t_{(h,j)} \notin \mathbf{s}^t_{(i,j)} \vee \mathbf{s}^t_{(i,j)} \in \left\{ \mathbf{r}^t_{(h,j)} - h \right\},\ \forall (i,j) \in \mathrm{R_c} \tag{2.6}$$

In Fig. 2.3(a), $\mathbf{s}^t_{(i,j)}$ is the route sequence along the route $s^t_{(i,j)}$ of the target content (k-th popular content). Meanwhile, $\mathbf{r}^t_{(h,j)}$ is the route sequence along the route of more popular content (m-th popular content: $m < k$) having the same origin site t as the target content. Furthermore, site h represents the starting site of the route sequence $\mathbf{r}^t_{(h,j)}$ and $\left\{ \mathbf{r}^t_{(h,j)} - h \right\}$ represents the subsequence excluding the starting site h from the route sequence $\mathbf{r}^t_{(h,j)}$.

Furthermore, the same replicas should not be cached on the designed transmission route of the target content (k-th popular content). This constraint is illustrated in Fig. 2.3(b).

$$s^t_{(i,j)} + u_k \leq 1 \quad \forall (i,j) \in \mathrm{R_c},\ k \in \left\{ \mathbf{s}^t_{(i,j)} - i \right\} \tag{2.7}$$



(a) Hierarchical route (b) Cache location

Figure 2.3: Constraint conditions

2.3 Distributed Cache Mechanism

In a distributed caching framework such as CCN, it is often difficult to get the request distribution for all content items. Moreover, the optimization model shown in Sect. 2.2 lacks scalability for large-scale networks because it belongs to the class of *NP-complete* problems. Therefore, we presented a distributed mechanism in [17] to locally search for energy efficient cache locations of content (chunks) for the same network model as Fig. 2.2.

In the proposed mechanism, every *CN* automatically pre-designs a threshold of request rates of content using local information on each caching hierarchy before cache operation. Meanwhile during the cache operation, each *CN* measures request rates of content (initial *Interest*s of content) and judges whether or not to cache chunks of the content using the pre-designed threshold as follows.

- When the request rate of content measured by CN_i is higher than the threshold set in CN_i, chunks of the content are stored on CN_i.

- When the request rate of content measured by CN_i is lower than the threshold set in CN_i, chunks of the content aren't cached in CN_i.

2.3.1 Process Flow

The threshold design is autonomously executed by each *CN* based on the following process. Here we define a parent *CN*, children *CN*s and branch *CN*s as the node directly over a target *CN*, nodes

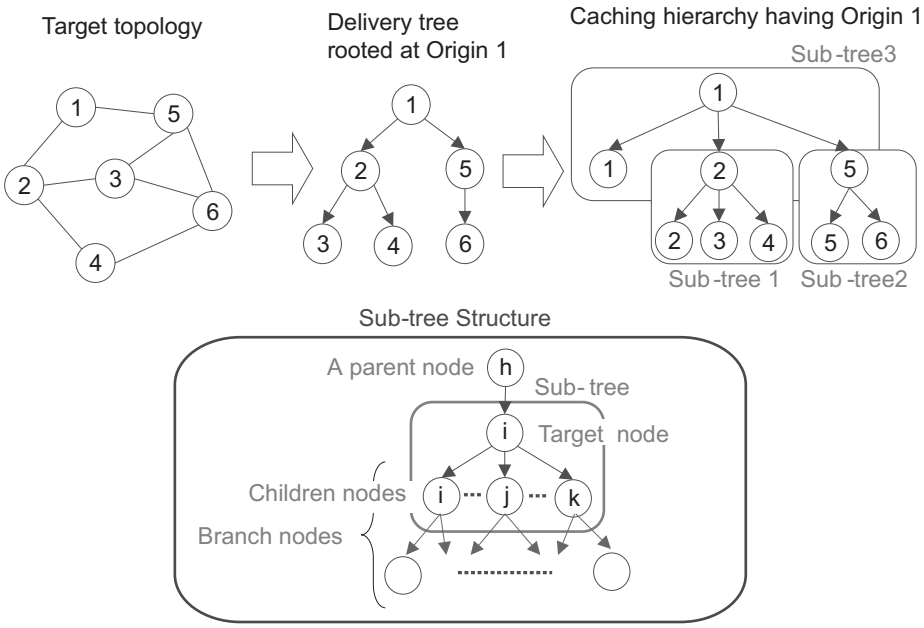


Figure 2.4: Example of creating a caching hierarchy and the sub-tree structure

directly under a target *CN* and all nodes below a target *CN* on a sub-tree in a caching hierarchy, respectively, as shown in Fig. 2.4.

For a caching hierarchy,

repeat

Target CN_i

step 1 model power consumption when a chunk is cached on CN_i and transmitted to all branch *CN*s.

step 2 design a threshold for each child *CN* using the power model and sets it to each child *CN* not including CN_i itself.

step 3 send the power model to parent CN_h

($\Rightarrow$ **step 1** for next target $CN_i \leftarrow CN_h$)

until CN_i is the root of the caching hierarchy.

Next, we show a distributed design algorithm of the threshold in Fig. 2.5.

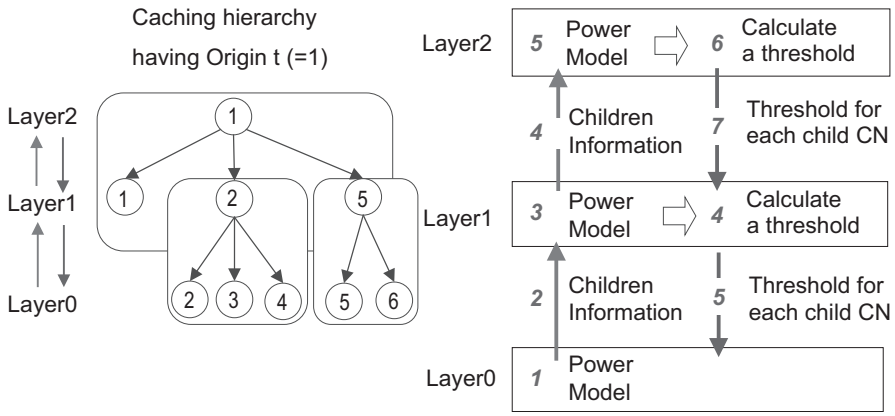


Figure 2.5: Bottom-up process

2.3.2 Distributed Algorithm for Threshold Design

We assume CN_i can know the following information from children *CN*s.

- Cr_i^t: the total number of children *CN*s for target CN_i on caching hierarchy t
- Br_i^t: the total number of branch *CN*s for target CN_i on caching hierarchy t
- H_i^t: the sum of hop-lengths from target CN_i to all branch *CN*s on caching hierarchy t

For content requested by each site at the rate of λ [requests/sec], the total power consumption when its chunk is cached on CN_i on caching hierarchy t and transmitted to all branch *CN*s is defined as

$$Power_i^t(\lambda) = Ca_i + Tr_i(\lambda, H_i^t), \quad \forall i. \tag{2.8}$$

Where the *cache allocation power* Ca_i [J] in 1 sec when a chunk is cached on CN_i is shown in Equation (2.2) and the *traffic transmission power* $Tr_i(\lambda, H_i^t)$ [J] when the chunk is transmitted from CN_i to all branch *CN*s is

$$Tr_i(\lambda, H_i^t) = D \cdot \lambda \cdot (P_r + P_{wdm}) \cdot H_i^t, \quad \forall i. \tag{2.9}$$

Next, CN_i designs a threshold using the power model. Under the assumption that content requests are symmetrically generated at all sites, CN_i designs a threshold by the following rule

for a sub-tree (cf. Fig. 2.4) using children information: $Power^t_j(\lambda)$, ($\forall j \in \mathbf{C^t_i}$: a set of children *CN*s).

if "$Power^t_i(\lambda)$: *the power consumption when a chunk having request rate* λ *from each site is cached only on* CN_i *delivering it to all branch CNs*" is greater than "$\sum_{j\in\mathbf{C^t_i}} Power^t_j(\lambda)$: *the power consumption when a chunk having request rate* λ *from each site is cached on* CN_j ($\forall j \in \mathbf{C^t_i}$) *delivering it to all branch CNs*" **then**

the chunk is cached on child CN_j ($\forall j \in \mathbf{C^t_i}$)

else

the chunk isn't cached on child CN_j ($\forall j \in \mathbf{C^t_i}$)

end if

CN_i calculates the boundary condition of the above-mentioned rule as follows.

$$Power^t_i(\lambda^t_b) = \sum_{j\in\mathbf{C^t_i}} Power^t_j(\lambda^t_b), \tag{2.10}$$

which leads to

$$\lambda^t_b = \frac{(Cr^t_i - 1)\cdot D \cdot P_{ca}}{D\cdot(P_r + P_{wdm})\cdot(H^t_i - \sum_{j\in\mathbf{C^t_i}} H^t_j)} \tag{2.11}$$

Therefore, the threshold Th^t_j of CN_j ($j \in \mathbf{C^t_i}$) in caching hierarchy t can be derived as

$$Th^t_j = Br^t_j \lambda^t_b. \tag{2.12}$$

After designing it, CN_i sets the threshold Th^t_j to child CN_j not including CN_i itself and the cache management in each *CN* is executed as follows.

For chunks of content having origin t,

- when the request rate of a content (initial *Interest*) measured by CN_j is higher than Th^t_j, chunks of the content are cached on the CN_j
- when the request rate of a content (initial *Interest*) measured by CN_j is lower than Th^t_j, chunks of the content aren't cached on the CN_j

Table 2.2: Variables in the distributed cache mechanism

Variable	Type of Variable	Definition
P_{ca}, P_r, P_{wdm}, D	Given	cf. Table 2.1
λ_b^t	Design	The boundary condition of the request rate of content generated from each site to calculate the threthold Th_j^t
Th_j^t	Design	Threshold of the request rate for CN_j in caching hierarchy t
Cr_j^t	Measure	The total number of children *CN*s of CN_i on caching hierarchy t
Br_j^t	Measure	The total number of branch *CN*s in a delivery tree rooted at CN_i on caching hierarchy t
H_j^t	Measure	The sum of hop-length from CN_i to all branch *CN*s.

Table 2.3: Power density parameters.

Device (Product)	Power	Spec	Power Density
DRAM	10 W	4 GB	$P_{ca} = 3.125 \times 10^{-10}\,\mathrm{J/(bit \cdot s)}$
Content Router (CRS-1)	4185 W	320 Gbps	$P_r = 1.3 \times 10^{-8}\,\mathrm{J/bit}$
WDM(FLASHWAVE9500)	800 W	480 Gbps	$P_{wdm} = 1.67 \times 10^{-9}\,\mathrm{J/bit}$

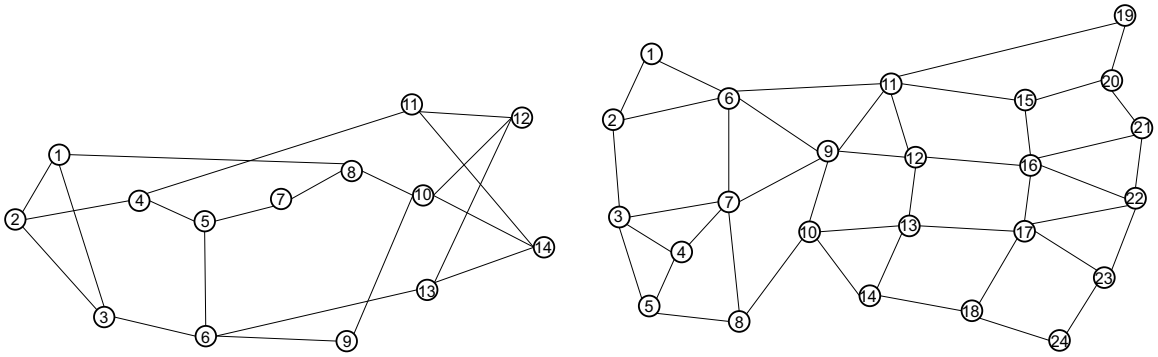
Furthermore after calculating the power model such as Equation (2.8), each *CN* sends it to its parent *CN*. Here, variables used in the threshold design are summarized in Table 2.2.

2.4 Evaluation

We first verified the tradeoff between cache allocation power and traffic transmission power for a chunk of content using the optimization model and evaluated the effectiveness of the distributed cache mechanism.

The simulation conditions are set to the following.

- **Test networks:** NSF topology with 14 *CN*s (Topology A), cf. Fig. 2.6(a) / US-backbone topology with 24 *CN*s (Topology B), cf. Fig. 2.6(b).
- **Content information:** Zipf-distributed requests from each site j for $K = 10000$ content items are defined as $R_{k,j} = \lambda = rk^{-\alpha}/c$, $c = \sum_{k=1}^{K} k^{-\alpha}$, cf. Fig. 2.7. We set α to 0.8 for UGC (User



(a) Topology A (b) Topology B

Figure 2.6: Test topologies

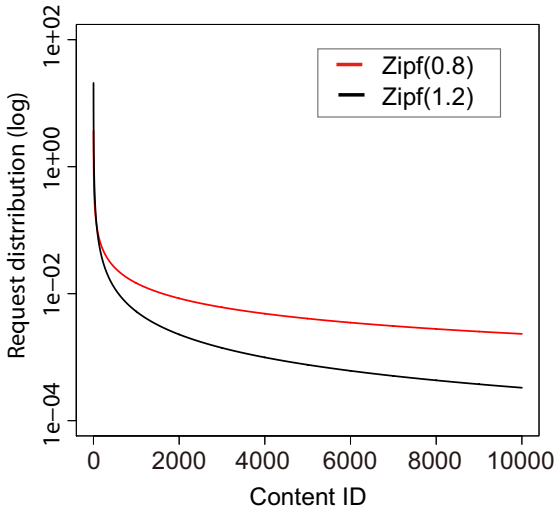
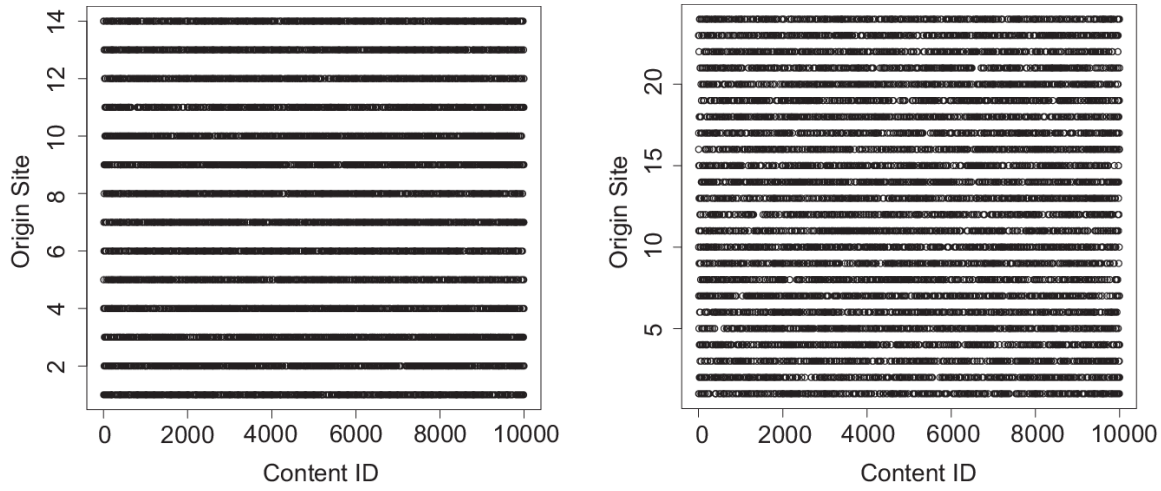


Figure 2.7: Request distribution of the content $R_{k,*}$

Generated Content) and 1.2 for VoD [42] and r to 100. Furthermore, the origin site t of content ID k is set randomly based on a uniform distribution, cf. Fig.s 2.8(a) and (b). The chunk size D and the average content size are set to 10 KB and 10 MB [43, 26]. The number of chunks n_k of content ID k follows a geometric distribution $\frac{1}{\sigma}(1-(\frac{1}{\sigma}))^{n_k}$ with the average number of chunks $\sigma = 1000 = 10\text{MB}/10\text{KB}$.

- **Power density of each device:** The power density of a memory device [J/(bit·s)] and a *CN*



(a) Origin sites of content in Topology A

(b) Origin sites of content in Topology B

Figure 2.8: Origin sites of content

or WDM node [J/bit] are set to the values given in Table 2.3.

2.4.1 Tradeoff of Power Consumption

We now compare power consumption for a chunk of content from a Zipf distribution with $\alpha = 1.2$ when the chunk is allocated using the following cache allocation policies.

- **Cache allocation on 1 *CN*:** The replica of a chunk of content is stored in a *CN*on its origin site.
- **Cache allocation on all *CN*s:** The replicas of a chunk of content are stored in all *CN*s.
- **Optimal cache allocation:** The replicas of a chunk of content are stored in *CN*s designed by our optimization model.

Figs. 2.9 and 2.11 show the power consumption for a chunk of content in each topology. Furthermore, Figs. 2.10 and 2.12 present the snapshots of each caching policy for content ID:398 having the origin site 7 in Topology A and content ID:395 having the origin site 12 in Topology B.

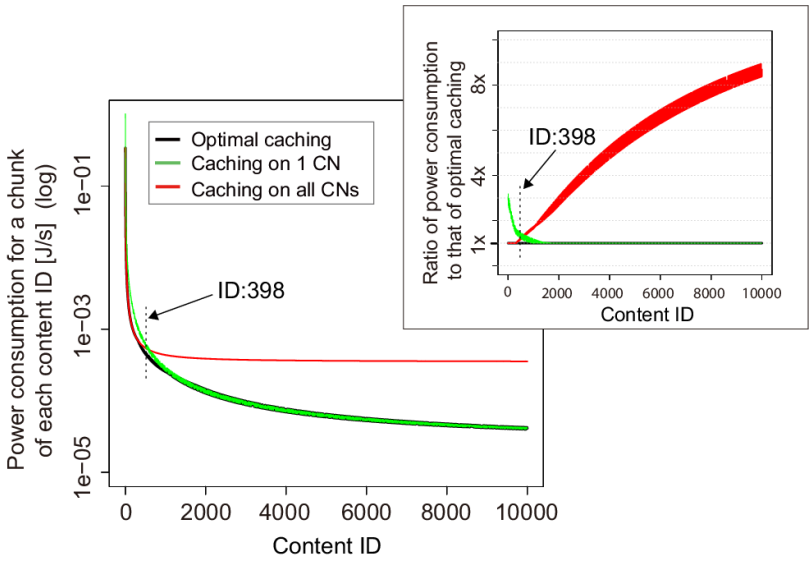


Figure 2.9: Power consumption for a chunk of each content ID and the ratio of power consumption to that of optimal caching (Zipf:$\alpha = 1.2$, TopologyA)

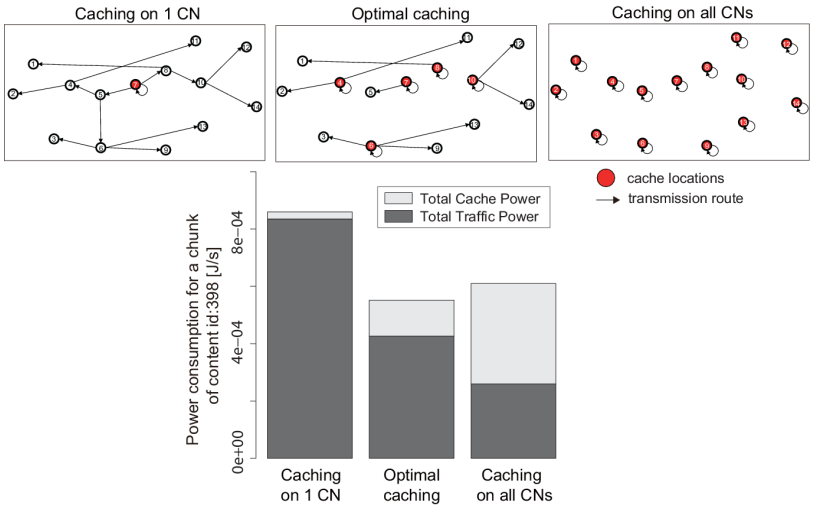


Figure 2.10: Power consumption for a chunk of content ID:398 (Zipf:$\alpha = 1.2$, TopologyA)

In these results, the cache locations for more popular content and less popular content in optimal caching are same as those in caching on all *CN*s and caching on 1 *CN*, respectively. However for content having intermediate popularity, the number of cache locations for a chunk in optimal caching is different from that in the other policies according to its request rates. Moreover, Figs

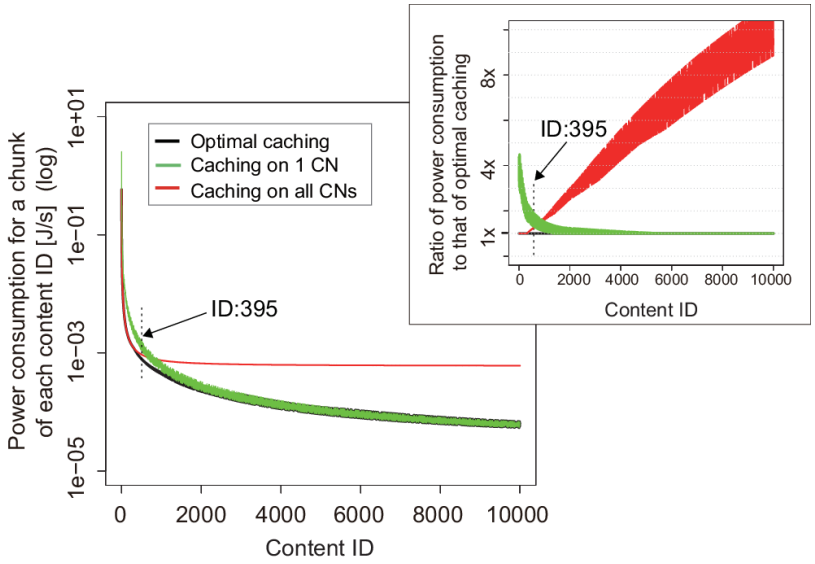


Figure 2.11: Power consumption for a chunk of each content ID and the ratio of power consumption to that of optimal caching (Zipf:$\alpha = 1.2$, TopologyB)

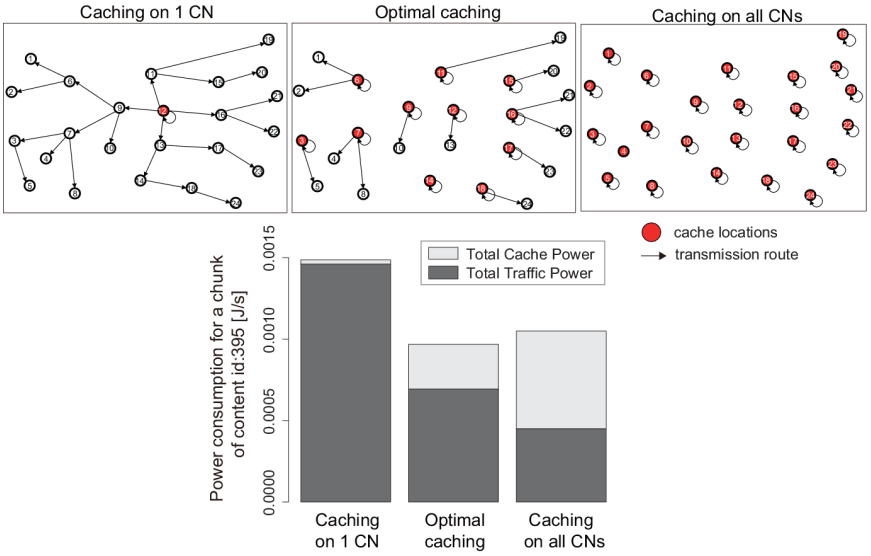


Figure 2.12: Power consumption for a chunk of content ID:395 (Zipf:$\alpha = 1.2$, TopologyB)

2.10 and 2.12 show the tradeoff relationship between the cache allocation power and the traffic transmission power and different content delivery topologies are derived from each cache allocation strategy. As a result, we see that the optimal cache allocation can derive the most energy efficient

locations for a chunk of content by balancing between the cache allocation power and the traffic transmission power according to the content popularity.

Therefore, distributed cache mechanisms for CCN should consider the tradeoff between the cache allocation power and the traffic transmission power and search for efficient cache locations of chunks of content by balancing the tradeoff on the caching hierarchies to be close to the optimal cache locations. Next, we demonstrate the effectiveness of our distributed cache mechanism.

2.4.2 Effectiveness of the Distributed Cache Mechanism

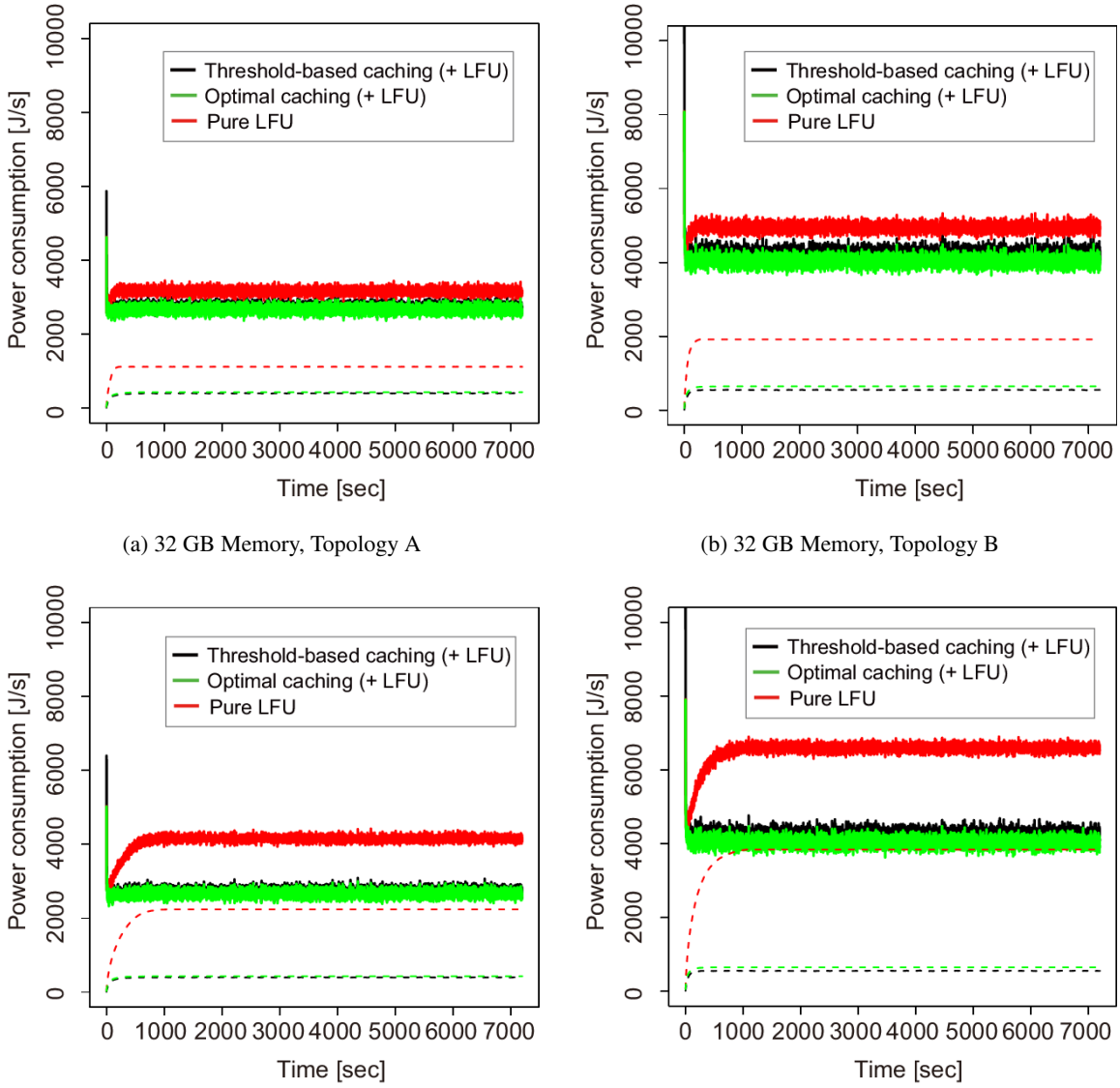
We evaluated the effectiveness of the distributed cache mechanism in chunk-level simulation and compared energy efficiency of three caching policies for a *CN* with different memory sizes (2, 4, 8, 12, 16, 20, 24, 32, 64, ∞ GB).

- **Optimal caching + LFU:** A chunk of content is cached on locations designed by the optimization model. When memory usage in a *CN* is above 100%, the chunk having the lowest request rate is discarded from memory of the *CN* according to LFU.

- **Threshold-based caching + LFU:** A chunk of content is cached when the request rate of the content is above a pre-designed threshold / isn't cached when the request rate is below the pre-designed threshold. Moreover, when memory usage in a *CN* is above 100%, the chunk having the lowest request rate is discarded from memory of the *CN* according to LFU.

- **Pure LFU:** A chunk of content is cached in all *CN*s thorough which it passes. Only when memory usage in a *CN* is above 100%, the chunk having the lowest request rate is discarded from memory of the *CN* according to LFU.

In the simulations, the generation of each request follows an exponential distribution and all *CN*s measure request rates of initial *Interests* of content for Pure LFU and threshold-based caching by using the exponentially weighted moving average (EWMA). The simulation time is set to 7200 sec.



(a) 32 GB Memory, Topology A

(b) 32 GB Memory, Topology B

(c) 64 GB Memory, Topology A

(d) 64 GB Memory, Topology B

Figure 2.13: Example of time change of total power consumption for content following a Zipf distribution with $\alpha = 1.2$ (solid line: total power consumption / dashed line: cache allocation power in the total power consumption)

2.4.3 Power Consumption and Memory Usage

We now compare the three caching policies in view of the total power consumption for two topologies. As examples, Fig. 2.13 shows the time change of total power consumption [J/s], which is

the sum of traffic transmission power and cache allocation power, when the memory size of each *CN* is set to 32 GB and 64 GB. Additionally, Fig. 2.14 presents differences of the average power consumption, which is the average of total power consumption in the steady state of the simulation when the memory size is changed. These results shown in Figs. 2.14(a)-(d) demonstrate that the power consumption in threshold-based caching is only slightly larger than that in optimal caching and lower than that in Pure-LFU.

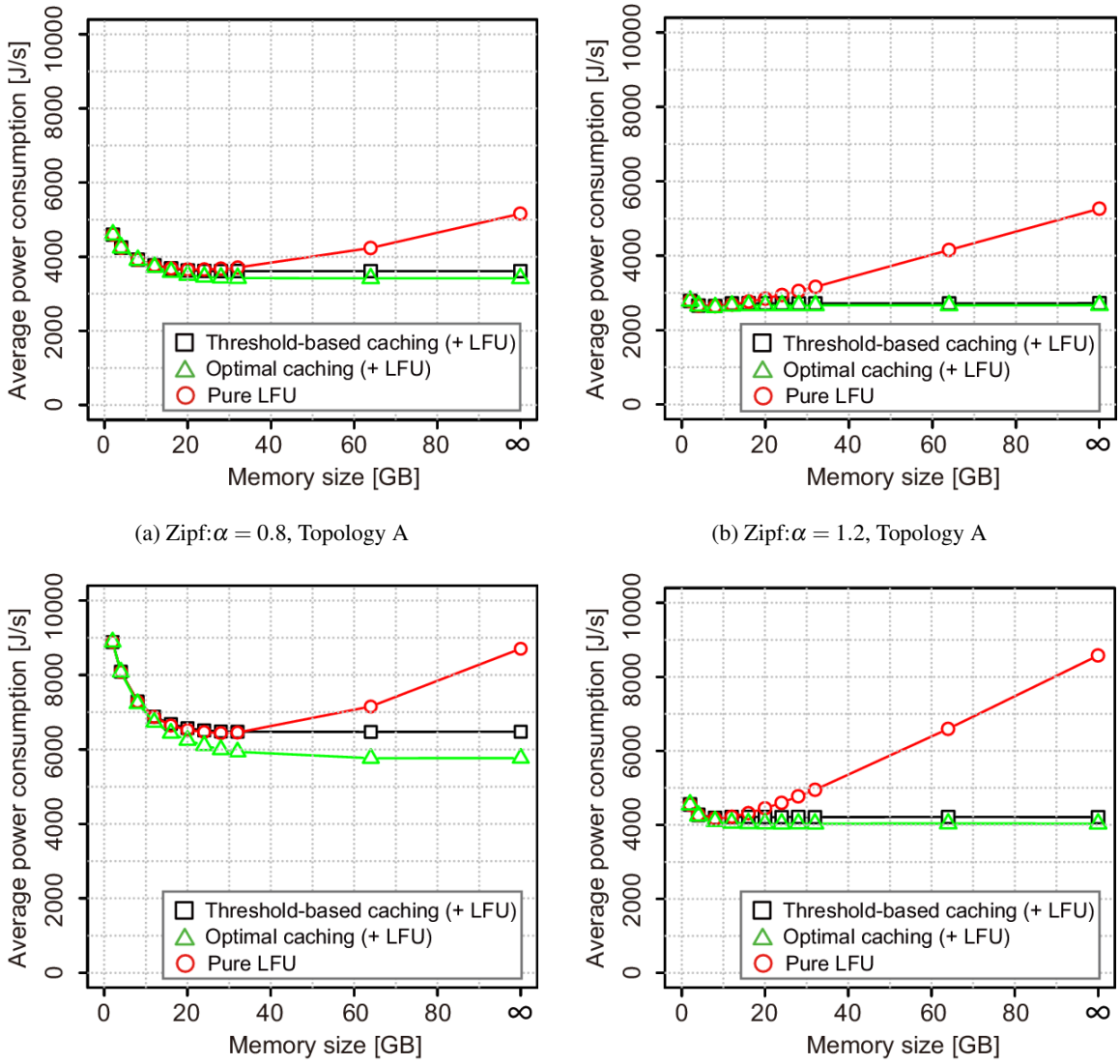
On the other hand, Fig. 2.14(c) shows larger differences of power consumption between threshold-based caching and optimal caching compared with the other results. In Fig. 2.15, we additionally show the total power consumption cumulated in the order from more popular content to less popular content for each topology which is composed of *CN*s having infinite memory. As shown in Fig. 2.15(c), the cumulative differences of power consumption between threshold-based caching and optimal caching are small for more popular content with IDs below 2000, but the cumulative power consumption shows larger differences for less popular content with IDs over 2000, in the long tailed distribution. That is why less popular content with $\alpha = 0.8$ has higher request rates and larger difference of power consumption than less popular content with $\alpha = 1.2$.

Furthermore in Fig. 2.16, we demonstrate the total power consumption for Topology B when the memory size of each *CN* is infinite and the request distribution is changed. For the Zipf-distributed requests from each site as $R_{k,j} = rk^{-\alpha}/c$, we changed α to $\{0.5, 0.8, 1.2, 1.5\}$ and r to $\{50, 100, 150\}$, respectively, as shown in Fig. 2.17.

As a result, we see that the differences between threshold-based caching and optimal caching are small when α is large or r is small, that is, less popular content has lower request rates. Meanwhile, energy efficiency in threshold-based caching is close to that in Pure-LFU as α becomes smaller or r becomes larger.

Moreover in these results, the power consumption in threshold-based caching for content with $\alpha = 1.2$ is smaller than that for content with $\alpha = 0.8$ because there are many contents having the request rates over the threshold for content with $\alpha = 0.8$, which discloses that the memory usage for content with $\alpha = 0.8$ is larger than that for content with $\alpha = 1.2$ in Fig. 2.18.

In Fig. 2.18, the total power consumption in threshold-based caching and optimal caching



(a) Zipf:$\alpha = 0.8$, Topology A

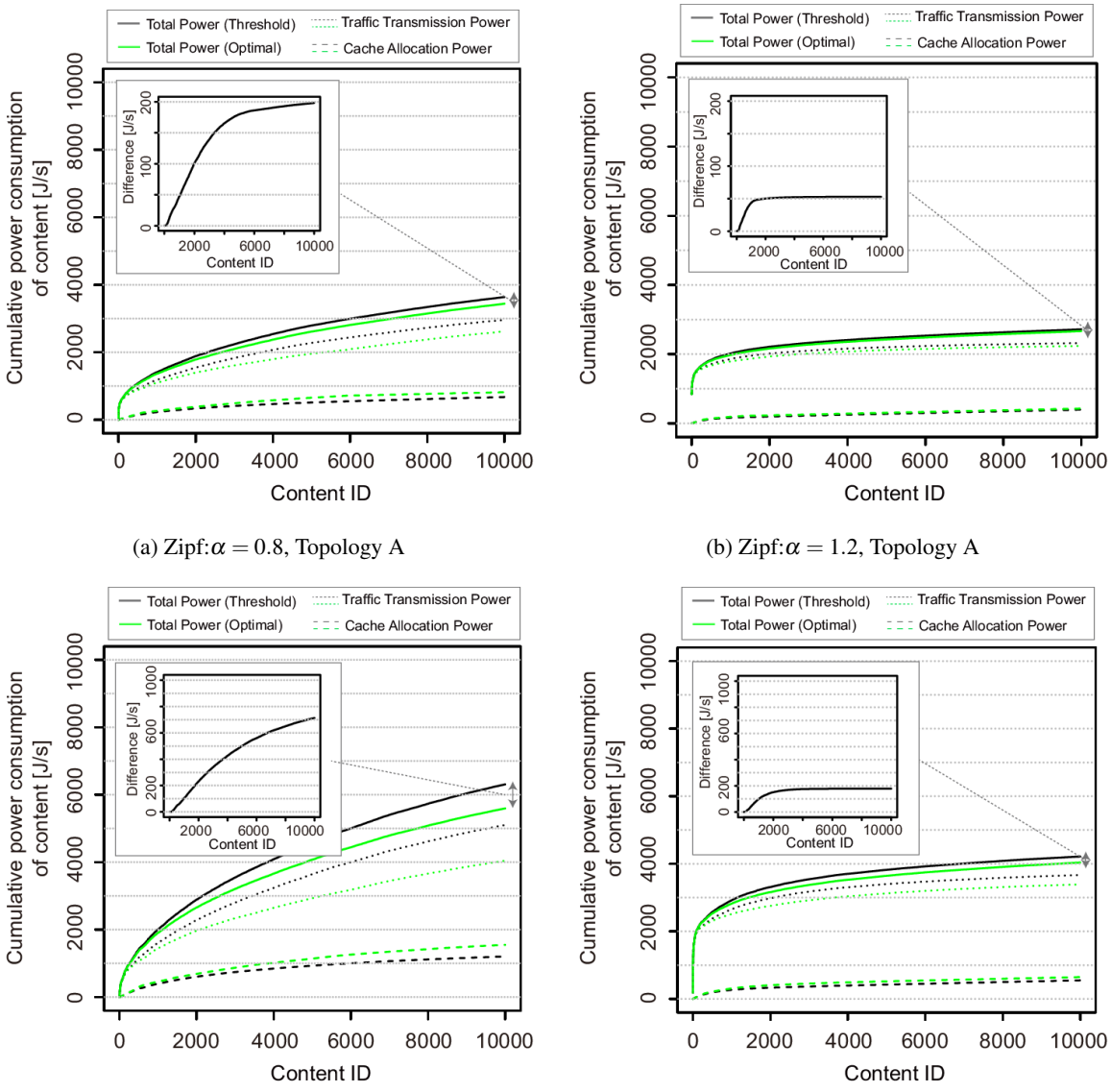
(b) Zipf:$\alpha = 1.2$, Topology A

(c) Zipf:$\alpha = 0.8$, Topology B

(d) Zipf:$\alpha = 1.2$, Topology B

Figure 2.14: Average power consumption when the memory size is changed

doesn't change even when the memory size is large, because the memory usage converges on the appropriate usage in view of energy efficiency. Meanwhile, the total power consumption in Pure LFU increases as the memory size is larger. This is why the cache allocation power becomes dominant in the total power consumption according to increasing memory size.



(a) Zipf:$\alpha = 0.8$, Topology A

(b) Zipf:$\alpha = 1.2$, Topology A

(c) Zipf:$\alpha = 0.8$, Topology B

(d) Zipf:$\alpha = 1.2$, Topology B

Figure 2.15: The cumulative power consumption of chunks of content for threshold-based caching and optimal caching when the memory size is infinite

As a result, we can see that the threshold-based caching using local searching realizes more energy efficient caching than Pure LFU and is near to the optimal solution using the whole network information. Meanwhile, these results show that the energy efficiency highly depends on the distribution of content popularity.

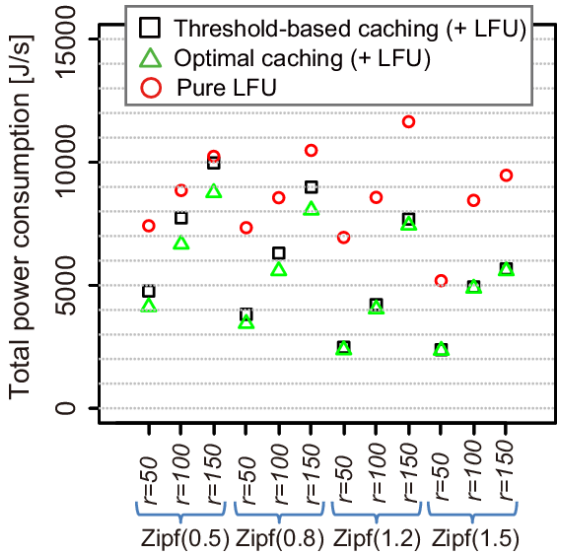


Figure 2.16: The total power consumption when the memory size is infinite and the request distribution is changed in Topology B

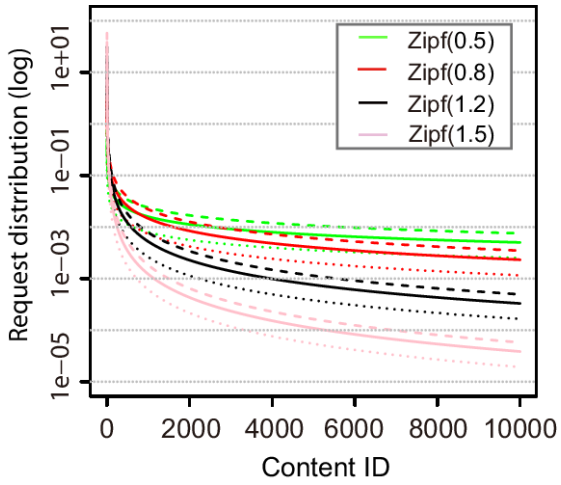
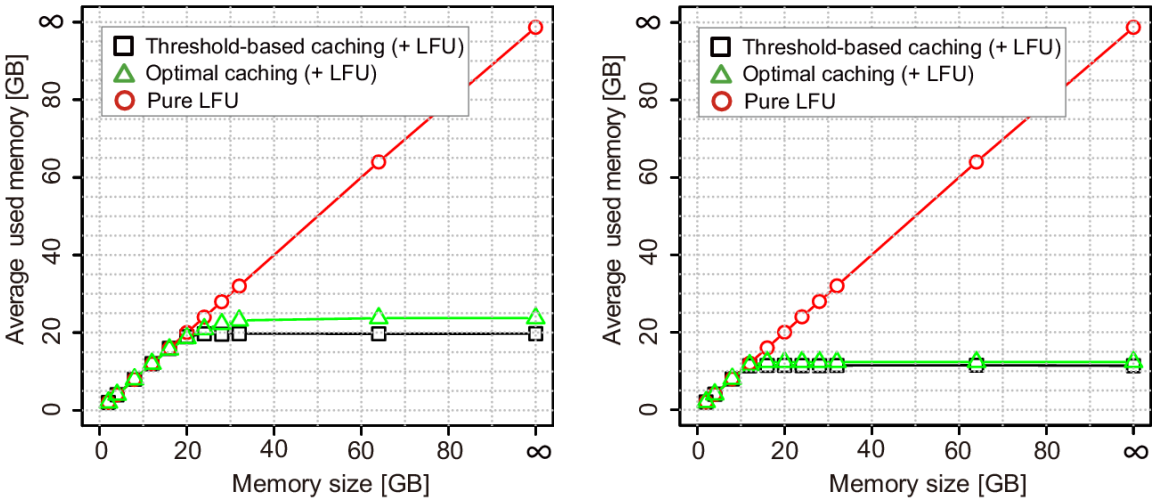


Figure 2.17: Request distribution of the content (dasched line: $r = 150$, solid line: $r = 100$, dotted line: $r = 50$)

2.4.4 Cache Performance

We next analyze the cache hit ratio defined as the ratio of chunks cached in the network for each request. Fig. 2.19 shows the average cache hit ratio for chunks of all content items when the memory



(a) Zipf:$\alpha = 0.8$, Topology A

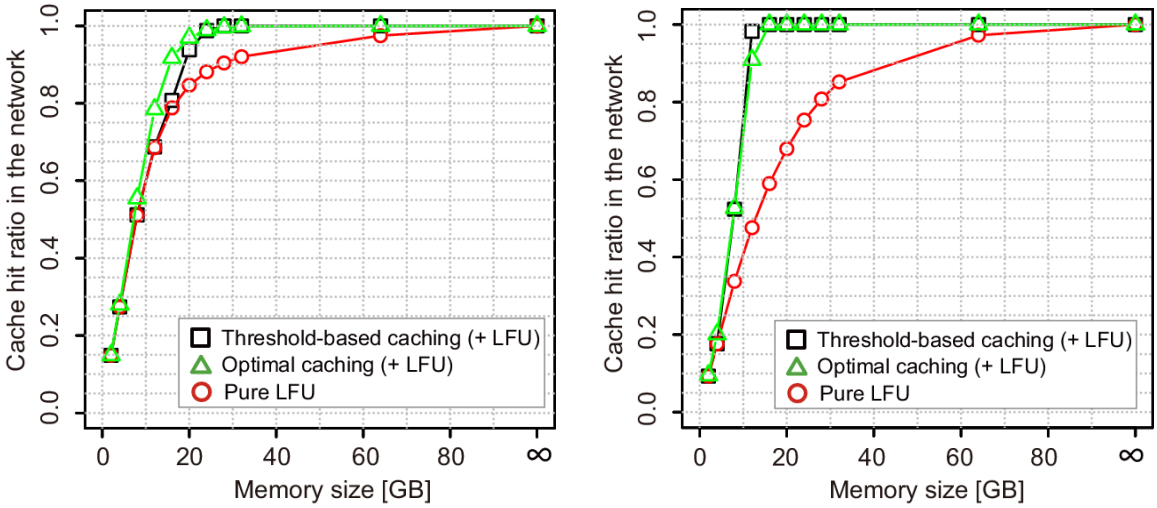
(b) Zipf:$\alpha = 1.2$, Topology A

(c) Zipf:$\alpha = 0.8$, Topology B

(d) Zipf:$\alpha = 1.2$, Topology B

Figure 2.18: Average used memory when the memory size is changed

size of each *CN* is changed. For content with $\alpha = 0.8$ in Figs. 2.19(a) and (c), the cache hit ratio in threshold-based caching is near to that in Pure LFU when the memory size is small and that in optimal caching only when the memory size is larger. For content with $\alpha = 1.2$ in Figures 2.19(b) and (d), the cache hit ratio in threshold-based caching is almost same as that in optimal caching even when the memory size is small. Meanwhile in all cases, the cache hit ratio in Pure LFU is



(a) Zipf:$\alpha = 0.8$, Topology A

(b) Zipf:$\alpha = 1.2$, Topology A

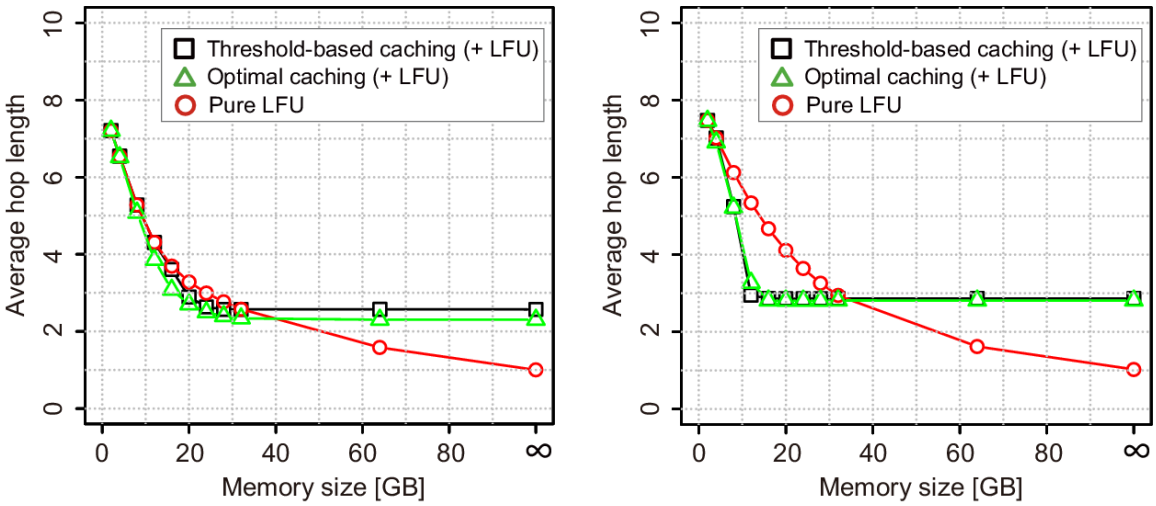
(c) Zipf:$\alpha = 0.8$, Topology B

(d) Zipf:$\alpha = 1.2$, Topology B

Figure 2.19: Cache hit ratio in the network when the memory size is changed

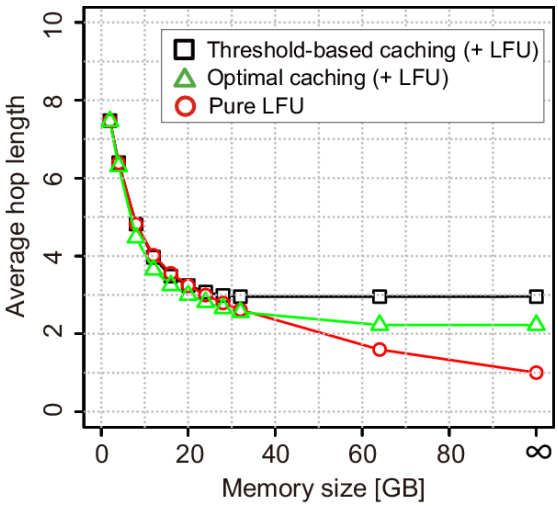
lower than that in the other policies.

Furthermore, the cache hit ratio in threshold-based caching for content with $\alpha = 0.8$ is higher than that for content with $\alpha = 1.2$ because less popular content with $\alpha = 0.8$ has higher request rates and is more easily cached in each *CN* than content with $\alpha = 1.2$. Although the cache hit ratio

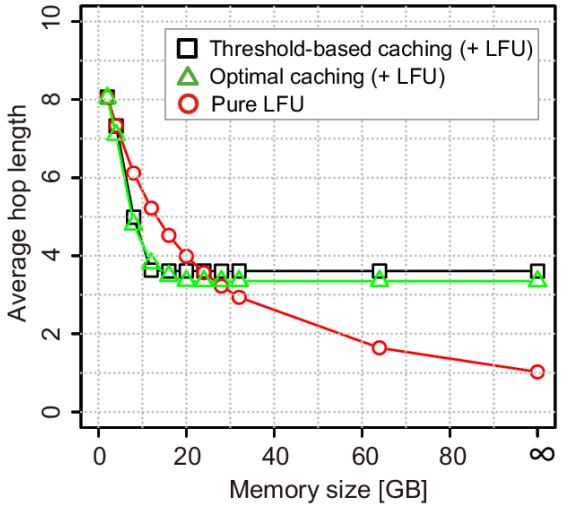


(a) Zipf:$\alpha = 0.8$, Topology A

(b) Zipf:$\alpha = 1.2$, Topology A



(c) Zipf:$\alpha = 0.8$, Topology B



(d) Zipf:$\alpha = 1.2$, Topology B

Figure 2.20: Average hop length when the memory size is changed

also depends on the distribution of content popularity, we see that the threshold-based caching can achieve good performance near to the optimal caching and improve not only the power consumption but also the cache hit ratio by effectively using network resources such as memory usage and bandwidth.

Additionally, Fig. 2.20 shows the average hop length for chunks of all content items when the

memory size of each *CN* is changed. For evaluation, we add a penalty of +5 to the hop length of content which isn't cached on any *CN*s in the network and for which a request (*Interest*) reaches its origin server. Figs. 2.20(b) and (d) illustrate that the average hop length in threshold-based caching is near to that in optimal caching. Furthermore the average hop-length in Pure LFU is longer than the others when the memory size of each *CN* is small, because many chunks of less popular content aren't cached in the network due to insufficient memory and many requests of content reach the origin server. Meanwhile, Figs. 2.20(a) and (c) show the average hop-length in optimal caching and threshold-based caching is close to that in Pure-LFU when the memory size is small. This is why chunks of many contents are allocated on many *CN*s as with Pure LFU. Moreover as the memory size is larger, the average hop length in Pure LFU becomes shorter than the others. Furthermore, the average hop length in threshold-based caching is longer than that in optimal caching because threshold-based caching discards more data than optimal caching as shown in Figs. 2.18(a) and (c).

Although the hop length also depends on the distribution of content popularity, we can see that the threshold-based caching can control content locations to be close to the optimal caching in consideration of the tradeoff between hop length i.e., response performance, and power consumption of the network.

2.5 Summary

In this chapter, we introduced an energy efficient design method to derive the optimal cache locations of chunks of content in order to provide reference locations to evaluate energy efficiency for cache strategies, which can consider the tradeoff between the cache allocation power and traffic transmission power under the constraints of the caching hierarchy. Furthermore, we proposed a distributed cache mechanism to locally search for energy efficient cache locations of chunks of content. In the mechanism, each *CN* pre-designs a threshold of request rates of chunks for each caching hierarchy and judges whether or not to cache the chunks by comparing measured request rates with the threshold.

In the simulation, we revealed the tradeoff between the cache allocation power and the traffic power for a chunk of content having different request rates and demonstrated that the proposed

distributed caching is near to the optimal solution derived by the proposed optimization model and can improve the total power consumption and the cache hit rate in the target network compared with Pure LFU. Furthermore, we showed that the energy efficiency of the proposed method depends on the distribution of content popularity.

Chapter 3

Modeling and Evaluation of Node-Level Cache Management

3.1 Issue and Approach for Analysis of Node-Level Cache Management

In caching systems, cache performance depends on the memory size at each node and the request distribution of content and it becomes an important issue to analyze the performance of current caching mechanisms for designing efficient cache systems.

Traditionally, statistical models for general caching mechanisms, such as LRU and FIFO replacement policies, have been well studied [23, 11, 44]. Beside the approximation models for LRU and FIFO policies, there have been some analysis models [45, 46, 47, 48, 49, 50] to design efficient caching systems connecting multiple queues in a single node or a caching network.

Feldman *et al.* [45] propose a multi-level LRU caching and construct an approximation model with multiple priority queues for service differentiation. The proposed mechanism uses LRU policy for partitioned multi-level queues. Gallo *et al.* [46] model cache performance for random and LRU policies in a single node and hierarchical caching systems. Furthermore, they also propose a tandem caching model in which random and LRU policies are sequentially connected and evaluate the caching performance by using the proposed models and simulations. Furthermore, Rosensweig

et al. [47] also propose an algorithm (*a-NET*) that approximates the cache performance of multi-caching networks, where nodes execute the LRU or random policies. The approximation models are constructed by Markov chains and can provide a good approximation to estimate cache performance in caching networks.

As a representative model for general caching mechanisms, the Che approximation [11, 34] can provide a highly accurate approximation of cache hit probability in LRU by using a simple model. Under stationary content popularity $\lambda(n)$, corresponding to the normalized access probability of object n to the total accesses $\sum_{n=1}^{N}\lambda(n)$ for N objects, the cache hit probability of object n for memory size C is derived as

$$H_{LRU}(n) = 1 - e^{-\lambda(n)\tau_C}, \tag{3.1}$$

where the characteristic time τ_C satisfies the condition of Eq. (3.2).

$$C = \sum_{n=1}^{N} 1 - e^{-\lambda(n)\tau_C} \tag{3.2}$$

Additionally, [34] also proposes an approximation model of cache hit probability under FIFO policy as

$$H_{FIFO}(n) = \frac{\lambda(n)t_C}{\sum_{i\neq n}\lambda(i) + \lambda(n)t_c} \tag{3.3}$$

where the characteristic time t_C satisfies the condition of Eq. (3.4).

$$C = \sum_{n}^{N} \frac{\lambda(n)t_C}{\sum_{i\neq n}\lambda(i) + \lambda(n)t_C} \tag{3.4}$$

Both approximations are simple but have the same accuracy as the approximation models proposed by Dan *et al.* [23]. Furthermore, Fofack *et al.* [44] enhance the approximation models in [23] to evaluate cache performance in a caching network where nodes execute LRU and FIFO policies. These model-based analyses can be executed in a shorter time than simulation-based evaluations. While LRU and FIFO policies realize cache operation of $O(1)$ in a single queue, their hitting performance

deteriorates when there are many *one-timers*. This is why relatively popular objects requested many times may be pushed out from the single queue by *one-timers*.

Meanwhile as effective caching for *one-timers*, 2Q and ARC mechanisms have been introduced and their caching operations are of the same complexity as LRU. These mechanisms can avoid inefficient eviction of relatively popular objects caused by *one-timers* because they maintain separate queues for *one-timers*. To analyze cache performance of the 2Q mechanism, Tanaka *et al.* [50] propose an enhanced 2Q mechanism that dynamically controls optimal partitioning for two queues by using *interreference interval for purged objects* (II-PO). They further propose an analytical model for optimal partitioning in 2Q by using II-PO and realize better performance than LRU.

However, in the aforementioned proposals, the approximations for 2Q and ARC only discuss empirical results on the cache performance by implementing these mechanisms. Moreover, to the best of our knowledge the modeling and theoretical analysis of cache performance in these mechanisms has not been established yet. Actually, it is difficult to analyze the interaction between separate queues compared with analyzing a single queue like for LRU or FIFO. Furthermore, it is an important issue to analyze the influence of cache pollution due to *one-timers* shown in the realistic workloads of web proxies and VoD services [24, 27, 28, 26, 29]. In this chapter, we aim to analyze the interactions between the separate queues in 2Q and ARC and to generate new statistical models of these mechanisms.

The remainder of this chapter is organized as follows. We review the caching mechanisms 2Q and ARC that are efficient for *one-timers* in Section 3.2 and propose the statistical models in Section 3.3. Section 3.4 demonstrates the approximation accuracy of the proposed models by comparing cache hit probability measured by simulation and that estimated by the proposed models. Finally we summarize this chapter in Section 3.5.

3.2 Caching Mechanisms for One-Timers

In LRU or random policies, when there are many objects that are requested only once and then never again, frequently requested objects may have to be removed due to the limited memory size. On the other hand, the *one-timers* don't influence cache removal of relatively popular objects in

Figure 3.1: Queue structure of Simplified 2Q

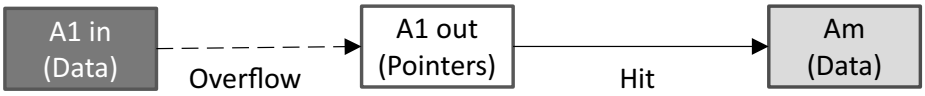


Figure 3.2: Queue structure of Full 2Q

mechanisms using separate queues for *one-timers*. As a consequence, these caching mechanisms can realize better performance than LRU and random policies when there are many one-timers.

Moreover to construct efficient cache systems in which nodes use 2Q or ARC, the operator should know the influence of node memory size and request distribution on cache performance in advance. However, it is difficult to analyze the cache characteristics because the multiple queues influence each other. Meanwhile to analyze the cache performance, simulations are useful, but require much more time for the evaluation. Therefore, this chapter aims at providing a design guideline for cache systems using 2Q or ARC and at modeling the statistical cache performance in the steady state for these caching mechanisms.

3.2.1 2Q

We first introduce the overall algorithms of the 2Q mechanism which has two variants named Simplified 2Q and Full 2Q [13]. As shown in Fig. 3.1, Simplified 2Q uses a combination of one FIFO queue A_1 and one LRU queue A_m. The A_1 queue is used to remember the access history of *one-timers*, which are generally just pointers named ghost caches. On the other hand, the A_m queue is used to cache popular objects that are requested many times while they are in the cache. For the first access of an object, Simplified 2Q caches the object data in the A_1 queue. If the same object is requested again while it is in the A_1 or A_m queue, the object will be moved to the tail of the A_m queue. When a new object is entered into one of the queues and the queue overflows, the oldest object at the head of the queue is removed. Simplified 2Q is known to work very well for stationary requests, but does not work well for nonstationary requests. Therefore, Full 2Q is proposed to

improve the nonstationary cache performance of Simplified 2Q. As shown in Fig. 3.2, the A_1 queue in Simplified 2Q is divided into two queues A_{1in} and A_{1out}. When an object is accessed for the first time, it is cached in the A_{1in} queue. Additionally, when the A_{1in} queue overflows and the oldest object at the head of the A_{1in} queue is pushed out, that object is stored as a ghost cache entry in the A_{1out} queue. If an object is hit in the A_{1out} or A_m queue, the object is moved to the tail of the A_m queue.

In 2Q, the sizes of all queues are constant. Therefore, we should design the partition sizes offline. Regarding Full 2Q, the sizes of A_{1in}, A_{1out}, and A_m queues are empirically set to 25% of the total memory size, the number of identifiers for as many objects as would fit on 50% of the memory size, and 75% of total memory size, respectively.

3.2.2 Adaptive Replacement Caching (ARC)

We next show the overall algorithms of ARC [14, 15]. ARC is an enhanced mechanism of *Fixed Replacement Caching* (FRC). ARC and FRC have two LRU lists called L_1 and L_2 as shown in Fig. 3.3. The L_1 list stores *one-timers* and the L_2 list stores objects accessed at least twice. Moreover, the L_1 list is divided into T_1 and B_1 queues which store object data and ghost caches, respectively. Similarly, the L_2 list is divided into T_2 and B_2. The sizes of the four queues satisfy the following conditions

$$0 \leq |L_1| + |L_2| \leq 2C, \quad 0 \leq |L_1| \leq C, \quad 0 \leq |L_2| \leq 2C$$

where the total memory size, which is composed of the T_1 and T_2 queues, is C. The symbol $|\cdot|$ denotes the total size of each list as the sum of the two queues for data and pointers. Furthermore if an object is requested again when it is already stored in T_2 or T_1 / remembered in B_1 or B_2, the object will be moved to the tail of the T_2 queue.

While FRC attempts to keep the sizes of the T_1 and T_2 queues as constant values p and $C - p$, ARC adaptively controls the tunable parameter p to track the sizes of the T_1 and T_2 queues as p and $C - p$ according to varying access patterns. Therefore, ARC behaves similarly to FRC except for the adaptively changing p.

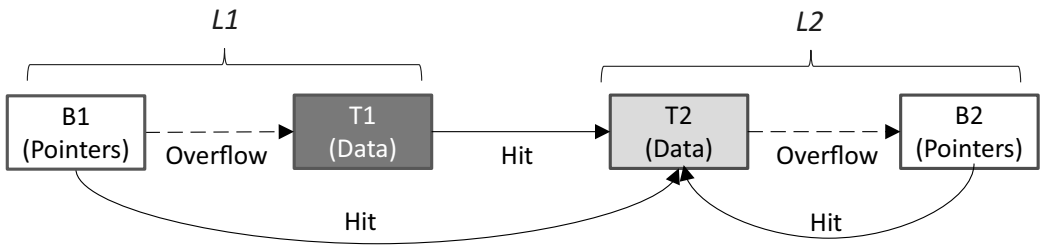


Figure 3.3: Queue structure of ARC

ARC automatically tunes the parameter p by predicting that the frequency of access to objects (i.e., *one-timers* or popular objects) will be increased/decreased in the near future as follows.

- When a request hits in B_1 (i.e., ghost caches for objects accessed once), ARC guesses that requests of other objects accessed once will also increase. As a result, the partition control of p is executed as follows.
  - If the size of T_1 $(= p)$ is small, i.e., *one-timers* have not been recently accessed, the size of T_1 is increased by 1
  - If the size of T_1 is large, the size of T_1 is increased by $p/(C-p)$
- When a request hits in B_2 (i.e., ghost caches for objects accessed more than once), ARC guesses that requests of objects accessed more than once will increase. As a result, the partition control of p is executed as follows.
  - If the size of T_2 $(= C-p)$ is small, i.e., objects referred more than once have not been recently accessed, the size of T_2 is increased by 1
  - If the size of T_2 is large, the size of T_2 is increased by $(C-p)/p$

In this algorithm of ARC, the cache hitting in B_1 suggests an increase in the size of T_1. Similarly, a cache hitting in B_2 suggests an increase in the size of T_2. For various access patterns, ARC can provide almost the same performance as that of FRC with optimal p tuned offline [15]. Moreover, the implementations of ARC and FRC also have low complexity because they are based on simple caching mechanisms such as LRU and FIFO.

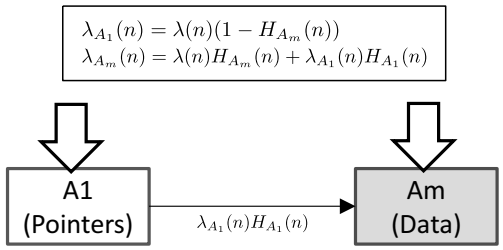


Figure 3.4: Queue model of Simplified 2Q

In this chapter, we consider approximating the cache characteristics of ARC by FRC with a given p and present the approximation accuracy in Sect. 3.4.

3.3 Proposed Approximation Models

We next propose new theorems for approximations of Simplified 2Q, Full 2Q, and FRC in Theorems 1, 2, and 3, respectively.

3.3.1 Approximation for Simplified 2Q

In consideration of the interaction between A_1 and A_m as shown in Fig. 3.4, the approximation model of Simplified 2Q can be derived as follows.

Theorem 1 *Under stationary popularity $\lambda(n)$ of object n, the cache hit probability of object n in Simplified 2Q is defined as follows.*

$$H_{S2Q}(n) = H_{A_m}(n). \tag{3.5}$$

The cache hit probability is derived by using the following hit probabilities $H_{A_1}(n)$, $H_{A_m}(n)$ in both queues.

- *Hit and request probabilities of object n in the A_1 queue*

$$H_{A_1}(n) = \frac{1-(1-\lambda_{A_1}(n))^{\tau_{A_1}}}{2}$$
$$\lambda_{A_1}(n) = \lambda(n)(1-H_{A_m}(n))$$

- *Hit and request probabilities of object n in the A_2 queue*

$$H_{A_m}(n) = 1 - e^{-\lambda_{A_m}(n)\tau_{A_m}}$$
$$\lambda_{A_m}(n) = \lambda(n)H_{A_m}(n) + \lambda_{A_1}(n)H_{A_1}(n)$$

The characteristic time of each queue, i.e., T_{A_1} and T_{A_2}, is obtained by solving

$$C_{A_1} = \sum_{n=1} H_{A_1}(n), \quad C_{A_m} = \sum_{n} H_{A_m}(n).$$

The parameters C_{A_1} and C_{A_m} are given as the sizes of the A_1 and A_m queues, respectively.

Proof 1 *For approximation of the A_1 queue, when $\bar{T}$ and C are the average queuing time of each object in microscopic time units and the size of the A_1 queue, the probability that object n is hit at the s-th reference slot after entering the A_1 queue is*

$$\text{Hit}(n) = \sum_{s=1}^{C\bar{T}} \lambda_{A_1}(n)(1-\lambda_{A_1}(n))^{(s-1)} = 1 - (1-\lambda_{A_1}(n))^{C\bar{T}}$$

Because the hit object in the A_1 queue moves to the A_m queue, the hit rate of object n in the A_1 queue is

$$H_{A_1}(n) = \frac{\text{Hit}(n)}{2} = \frac{1-(1-\lambda_{A_1}(n))^{C\bar{T}}}{2}.$$

Suppose that $C\bar{T}$ is the characteristic time τ_{A_1} of the A_1 queue, the hit probability of object n in A_1 queue is defined as

$$H_{A_1}(n) = \frac{1-(1-\lambda_{A_1}(n))^{\tau_{A_1}}}{2}.$$

Furthermore, the A_m queue can be proved in a similar way to the Che approximation [11]. □

The characteristic times τ_{A_1} and τ_{A_m} can be calculated by Algorithm 1 with an enhancement of the iterative calculation proposed in [44].

Algorithm 1 Calculation of Simplified 2Q model

Require: $\lambda(1), \cdots, \lambda(N), C_{A_1}, C_{A_m}$
Ensure: $H_{S2Q}(1), \cdots, H_{S2Q}(N)$
  Set τ_{A_1}, τ_{A_m}, $H_{A_1}(n)$, $H_{A_m}(n)$ to arbitrary initial values
  $B_{A_1} \leftarrow 0, \quad B_{A_m} \leftarrow 0$
  for $n \leftarrow 1, N$ **do**
    $B_{A_1} \leftarrow B_{A_1} + H_{A_1}(n)$
    $B_{A_m} \leftarrow B_{A_m} + H_{A_m}(n)$
  end for
  while $|C_{A_1} - B_{A_1}| \ll \delta \wedge |C_{A_m} - B_{A_m}| \ll \delta$ **do**
    $\{\delta$: the minimum value for precision$\}$
    $B_{A_1} \leftarrow 0$, $B_{A_m} \leftarrow 0$
    for $n \leftarrow 1, N$ **do**
      $\lambda_{A_1}(n) \leftarrow \lambda(n)(1 - H_{A_1}(n))(1 - H_{A_m}(n))$
      $\lambda_{A_m}(n) \leftarrow \lambda(n) H_{A_m}(n) + \lambda_{A_1}(n) H_{A_1}(n)$
      $H_{A_1}(n) \leftarrow \frac{1-(1-\lambda_{A_1}(n))^{\tau_{A_1}}}{2}$
      $H_{Am}(n) \leftarrow 1 - e^{-\lambda_{A_m}(n)\tau_{A_m}}$
      $B_{A_1} \leftarrow B_{A_1} + H_{A_1}(n)$
      $B_{A_m} \leftarrow B_{A_m} + H_{A_m}(n)$
      $\{$calculations for Theorem 1$\}$
    end for
    $\tau_{A_1} \leftarrow \tau_{A_1} + \alpha(C_{A_1} - B_{A_1})$
    $\tau_{A_m} \leftarrow \tau_{A_m} + \alpha(C_{A_m} - B_{A_m})$
    $\{$search for the characteristic time of each queue (α: a tunable parameter)$\}$
    for $n \leftarrow 1, N$ **do**
      $H_{S2Q}(n) \leftarrow H_{A_m}(n)$
    end for
  end while

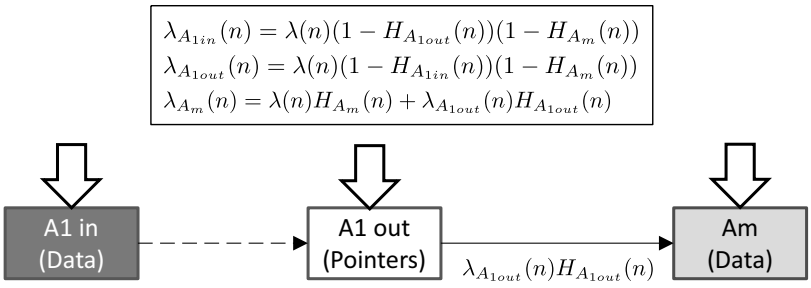


Figure 3.5: Queue model of Full 2Q

3.3.2 Approximation for Full 2Q

As shown in Fig. 3.5, the approximation model for the A_{1in}, A_{1out}, and A_m queues in Full 2Q can be defined as follows.

Theorem 2 *Under the stationary popularity $\lambda(n)$ of object n, the hit probability of object n in Full 2Q is defined as*

$$H_{F2Q}(n) = H_{A_{1in}}(n) + H_{A_m}(n) \tag{3.6}$$

by using the following hit probabilities $H_{A_{1in}}(n)$, $H_{A_{1out}}(n)$, and $H_{A_m}(n)$ of the three queues.

- *Hit and request probabilities of object n in the A_{1in} queue*

$$H_{A_{1in}}(n) = 1 - (1 - \lambda_{A_{1in}}(n))^{\tau_{A_{1in}}}$$
$$\lambda_{A_{1in}}(n) = \lambda(n)(1 - H_{A_{1out}}(n))(1 - H_{A_m}(n))$$

- *Hit and request probabilities of object n in the $A_{1_{out}}$ queue*

$$H_{A_{1out}}(n) = \frac{1 - (1 - \lambda_{A_{1out}}(n))^{\tau_{A_{1out}}}}{2}$$
$$\lambda_{A_{1out}}(n) = \lambda(n)(1 - H_{A_{1in}}(n))(1 - H_{A_m}(n))$$

- *Hit and request probabilities of object n in the A_m queue*

$$H_{A_m}(n) = 1 - e^{-\lambda_{A_m}(n)\tau_{A_m}}$$
$$\lambda_{A_m}(n) = \lambda(n)H_{A_m}(n) + \lambda_{A_{1out}}(n)H_{A_{1out}}(n)$$

The characteristic time of each queue, i.e., $\tau_{A_{1in}}$, $\tau_{A_{1out}}$, and τ_{A_m}, is obtained by solving

$$C_{A_{1in}} = \sum_n H_{A_{1in}}(n), \quad C_{A_{1out}} = \sum_n H_{A_{1out}}(n), \quad C_{A_m} = \sum_n H_{A_m}(n).$$

These parameters $C_{A_{1in}}$, $C_{A_{1out}}$, and C_{A_m} are given as the sizes of the A_{1in}, A_{1out}, and A_m queues, respectively.

Proof 2 *For approximation of the A_{1in} queue, when $\bar{T}$, C are the average queuing time of each object in microscopic time units and the size of the A_{1in} queue, the probability that object n is hit at the s-th reference slot after entering the A_{1in} queue is*

$$H_{A_{1in}}(n) = \sum_{s=1}^{C\bar{T}} \lambda_{A_{1in}}(n)(1-\lambda_{A_{1in}}(n))^{(s-1)} = 1 - (1-\lambda_{A_{1in}}(n))^{C\bar{T}}.$$

Suppose that $C\bar{T}$ is the characteristic time $\tau_{A_{1in}}$ of the A_{1in} queue, the hit probability of object n in the A_{1in} queue is defined as

$$H_{A_{1in}}(n) = 1 - (1-\lambda_{A_{1in}}(n))^{\tau_{A_{1in}}}.$$

Furthermore, the A_{1out} and A_m queues can be proved in a similar way with the approximation for the A_1 and A_m queues in Simplified 2Q. □

The characteristic times $\tau_{A_{1in}}$, $\tau_{A_{1out}}$, and τ_{A_2} can be calculated by Algorithm 2.

Algorithm 2 Calculation of Full 2Q model

Require: $\lambda(1), \cdots, \lambda(N), C_{A_{1in}}, C_{A_{1out}}, C_{A_m}$
Ensure: $H_{F2Q}(1), \cdots, H_{F2Q}(N)$
  Set $\tau_{A_{1in}}$, $\tau_{A_{1out}}$, τ_{A_m}, $H_{A_{1in}}(n)$, $H_{A_{1out}}(n)$, $H_{A_m}(n)$ to arbitrary initial values
  $B_{A_{1in}} \leftarrow 0, \quad B_{A_{1out}} \leftarrow 0, \quad B_{A_m} \leftarrow 0$
  for $n \leftarrow 1, N$ **do**
    $B_{A_{1in}} \leftarrow B_{A_{1in}} + H_{A_{1in}}(n)$
    $B_{A_{1out}} \leftarrow B_{A_{1out}} + H_{A_{1out}}(n)$
    $B_{A_m} \leftarrow B_{A_m} + H_{A_m}(n)$
  end for
  while $|C_{A_{1in}} - B_{A_{1in}}| \ll \delta \wedge |C_{A_{1out}} - B_{A_{1out}}| \ll \delta \wedge |C_{A_m} - B_{A_m}| \ll \delta$ **do**
    $\{\delta$: the minimum value for precision$\}$
    $B_{A_{1in}} \leftarrow 0, \quad B_{A_{1out}} \leftarrow 0, \quad B_{A_m} \leftarrow 0$
    for $n \leftarrow 1, N$ **do**
      $\lambda_{A_1 in}(n) \leftarrow \lambda(n)(1 - H_{A_{1out}}(n))(1 - H_{A_m}(n))$
      $\lambda_{A_{1out}}(n) \leftarrow \lambda(n)(1 - H_{A_{1in}}(n))(1 - H_{A_m}(n))$
      $\lambda_{A_m}(n) \leftarrow \lambda(n) H_{A_m}(n) + \lambda_{A_{1out}}(n) H_{A_{1out}}(n)$
      $H_{A_{1in}}(n) \leftarrow 1 - (1 - \lambda_{A_{1in}}(n))^{\tau_{A_{1in}}}$
      $H_{A_{1out}}(n) \leftarrow \frac{1-(1-\lambda_{A_{1out}}(n))^{\tau_{A_{1out}}}}{2}$
      $H_{Am}(n) \leftarrow 1 - e^{-\lambda_{A_m}(n)\tau_{A_m}}$
      $B_{A_{1in}} \leftarrow B_{A_{1in}} + H_{A_{1in}}(n)$
      $B_{A_{1out}} \leftarrow B_{A_{1out}} + H_{A_{1out}}(n)$
      $B_{A_m} \leftarrow B_{A_m} + H_{A_m}(n)$
      $\{$calculations for Theorem 2$\}$
    end for
    $\tau_{A_{1in}} \leftarrow \tau_{A_{1in}} + \alpha(C_{A_{1in}} - B_{A_{1in}})$
    $\tau_{A_{1out}} \leftarrow \tau_{A_{1out}} + \alpha(C_{A_{1out}} - B_{A_{1out}})$
    $\tau_{A_m} \leftarrow \tau_{A_m} + \alpha(C_{A_m} - B_{A_m})$
    $\{$search for the characteristic time of each queue (α: a tunable parameter)$\}$
    for $n \leftarrow 1, N$ **do**
      $H_{F2Q}(n) \leftarrow H_{A_{1_{in}}}(n) + H_{A_m}(n)$
    end for
  end while

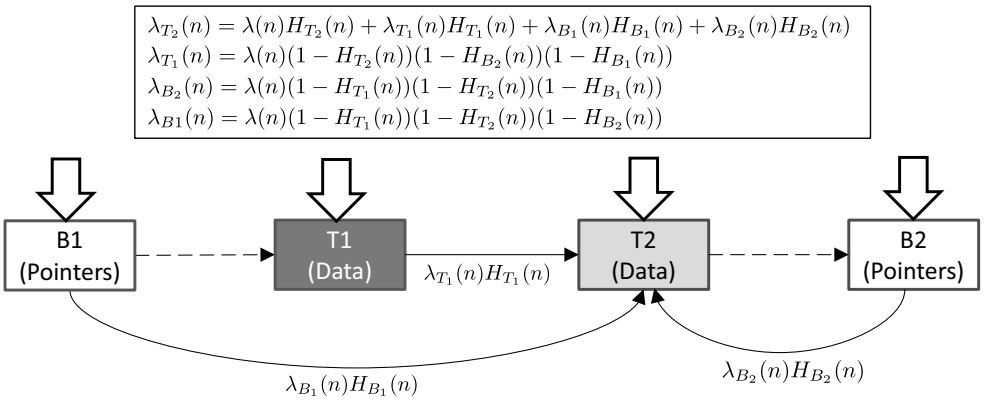


Figure 3.6: Queue model of ARC

3.3.3 Approximation for ARC

In this chapter, we analyze the cache characteristics of ARC by using FRC with a given and constant p. In consideration of the interaction between T_2, T_1, B_2, and B_1 as shown in Fig. 3.6, the approximation model of FRC can be defined as follows.

Theorem 3 *Under the stationary popularity $\lambda(n)$ of object n, the hit probability of object n in FRC with a given p is defined as*

$$H_{FRC}(n) = H_{T_1}(n) + H_{T_2}(n) \tag{3.7}$$

by using cache hit probability $H_{T_1}(n)$, $H_{T_2}(n)$, $H_{B_1}(n)$ and $H_{B_2}(n)$ in the four queues:

- *Hit and request probabilities of object n in the T_2 queue*

$$H_{T_2}(n) = 1 - e^{-\lambda_{T_2}(n)\tau_{T_2}}$$
$$\lambda_{T_2}(n) = \lambda(n)H_{T_2}(n) + \lambda_{T_1}(n)H_{T_1}(n) + \lambda_{B_1}(n)H_{B_1}(n) + \lambda_{B_2}(n)H_{B_2}(n)$$

- *Hit and request probabilities of object n in the T_1 queue*

$$H_{T_1}(n) = \frac{1-(1-\lambda_{T_1}(n))^{\tau_{T_1}}}{2}$$
$$\lambda_{T_1}(n) = \lambda(n)(1-H_{T_2}(n))(1-H_{B_2}(n))(1-H_{B_1}(n))$$

- *Hit and request probabilities of object n in the B_1 queue*

$$H_{B_1}(n) = \frac{1-(1-\lambda_{B_1}(n))^{\tau_{B_1}}}{2}$$
$$\lambda_{B_1}(n) = \lambda(n)(1-H_{T_1}(n))(1-H_{T_2}(n))(1-H_{B_2}(n))$$

- *Hit and request probabilities of object n in the B_2 queue*

$$H_{B_2}(n) = \frac{1-(1-\lambda_{B_2}(n))^{\tau_{B_1}}}{2}$$
$$\lambda_{B_2}(n) = \lambda(n)(1-H_{T_1}(n))(1-H_{T_2}(n))(1-H_{B_1}(n))$$

The characteristic time of each queue, i.e., τ_{T_1}, τ_{T_2}, τ_{B_1}, and τ_{B_2}, is obtained by solving

$$p = \sum_n H_{T_1}(n), \quad C-p = \sum_n H_{T_2}(n), \quad C-p = \sum_n H_{B_1}(n), \quad p = \sum_n H_{B_2}(n).$$

The parameters C and p are given as the total memory size and the size of the T_1 queue, respectively.

Proof 3 *The T_2 queue can be proved in a similar way to the Che approximation [11]. Furthermore, the T_1, B_1, and B_2 queues can be proved in a similar way with the approximation for the A_1 queue in Simplified 2Q.* □

The characteristic times τ_{T_2}, τ_{T_1}, τ_{B_1} and τ_{B_2} can be calculated by Algorithm 3.

Actually, in Sect. 3.4, we confirm that the size of the T_1 queue (i.e., p) in ARC always varies at small values for Zipf-like content popularity and the cache hit probability of ARC measured by simulations is almost equal to that estimated by the FRC(p) model with small p. In the following evaluations, we will use the FRC(1) model of setting $p = 1$ as the approximation of ARC.

Algorithm 3 Calculation of FRC model

Require: $p, \lambda(1), \cdots, \lambda(N)$
Ensure: $H_{FRC}(1), \cdots, H_{FRC}(N)$
  Set τ_{T_2}, τ_{T_1}, τ_{B_2}, τ_{B_1}, $H_{T_1}(n)$, $H_{T_2}(n)$, $H_{B_1}(n)$, $H_{B_2}(n)$ to arbitrary initial values
  $C_{T_2} \leftarrow C - p, \quad C_{T_1} \leftarrow p, \quad C_{B_1} \leftarrow p, \quad C_{B_2} \leftarrow C - p$
  $B_{T_2} \leftarrow 0, \quad B_{T_1} \leftarrow 0, \quad B_{B_2} \leftarrow 0, \quad B_{B_1} \leftarrow 0$
  for $n \leftarrow 1, N$ **do**
    $B_{T_2} \leftarrow B_{T_2} + H_{T_2}(n)$
    $B_{T_1} \leftarrow B_{T_1} + H_{T_1}(n)$
    $B_{B_2} \leftarrow B_{B_2} + H_{B_2}(n)$
    $B_{B_1} \leftarrow B_{B_1} + H_{B_1}(n)$
  end for
  while $|C_{T_2} - B_{T_2}| \ll \delta \wedge |C_{T_1} - B_{T_1}| \ll \delta \wedge |C_{B_2} - B_{B_2}| \ll \delta \wedge |C_{B_1} - B_{B_1}| \ll \delta$ **do**
    $\{\delta$: the minimum value for precision$\}$
    $B_{T_2} \leftarrow 0, B_{T_1} \leftarrow 0, B_{B_2} \leftarrow 0, B_{B_1} \leftarrow 0$
    for $n \leftarrow 1, N$ **do**
      $\lambda_{T_2}(n) \leftarrow \lambda(n) H_{T_2}(n) + \lambda_{T_1}(n) H_{T_1}(n) + \lambda_{B_1}(n) H_{B_1}(n) + \lambda_{B_2}(n) H_{B_2}(n)$
      $\lambda_{T_1}(n) \leftarrow \lambda(n)(1 - H_{T_2}(n))(1 - H_{B_2}(n))(1 - H_{B_1}(n))$
      $\lambda_{B_1}(n) \leftarrow \lambda(n)(1 - H_{T_1}(n))(1 - H_{T_2}(n))(1 - H_{B_2}(n))$
      $\lambda_{B_2}(n) \leftarrow \lambda(n)(1 - H_{T_1}(n))(1 - H_{T_2}(n))(1 - H_{B_1}(n))$
      $H_{T_2}(n) \leftarrow 1 - e^{-\lambda_{T_2}(n)\tau_{T_2}}$
      $H_{T_1}(n) \leftarrow \frac{1-(1-\lambda_{T_1}(n))^{\tau_{T_1}}}{2}$
      $H_{B_1}(n) \leftarrow \frac{1-(1-\lambda_{B_1}(n))^{\tau_{B_1}}}{2}$
      $H_{B_2}(n) \leftarrow \frac{1-(1-\lambda_{B_2}(n))^{\tau_{B_1}}}{2}$
      $B_{T_2} \leftarrow B_{T_2} + H_{T_2}(n)$
      $B_{T_1} \leftarrow B_{T_1} + H_{T_1}(n)$
      $B_{B_1} \leftarrow B_{B_1} + H_{B_1}(n)$
      $B_{B_2} \leftarrow B_{B_2} + H_{B_2}(n)$
      $\{$calculations for Theorem 3$\}$
    end for
    $\tau_{T_2} \leftarrow \tau_{T_2} + \alpha(C_{T_2} - B_{T_2})$
    $\tau_{T_1} \leftarrow \tau_{T_1} + \alpha(C_{T_1} - B_{T_1})$
    $\tau_{B_2} \leftarrow \tau_{B_2} + \alpha(C_{B_2} - B_{B_2})$
    $\tau_{B_1} \leftarrow \tau_{B_1} + \alpha(C_{B_1} - B_{B_1})$
    $\{$search for the characteristic time of each queue (α: a tunable parameter)$\}$
    for $n \leftarrow 1, N$ **do**
      $H_{FRC}(n) \leftarrow H_{T_1}(n) + H_{T_2}(n)$
    end for
  end while

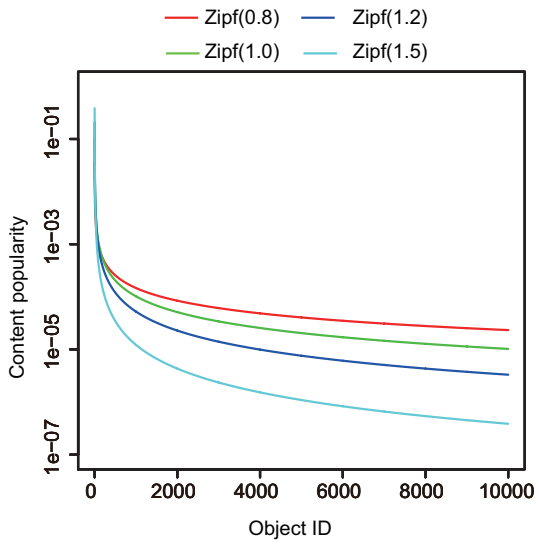


Figure 3.7: Content popularity

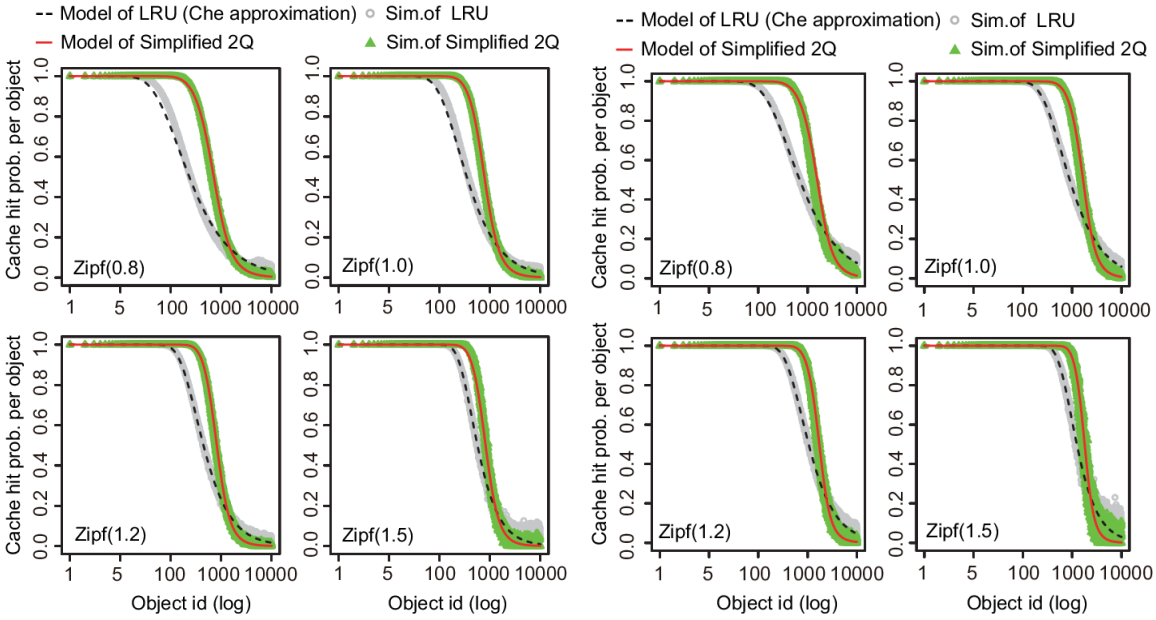
3.4 Evaluation

To confirm the approximation accuracy of the proposed models of 2Q and ARC, we compare the cache hit probability derived by Theorems 1-3 with that measured by simulations for content requests generated at exponentially distributed intervals. The evaluation conditions are set to the following.

- **Content information:** Zipf-distributed popularity of object n for $N = 10^4$ objects are defined as $\lambda(n) = k^{-\alpha}/c$, $c = \sum_{k=1}^{N} k^{-\alpha}$, cf. Fig. 3.7. We set α to 0.8, 1.0, 1.2, and 1.5 as realistic parameters for User Generated Content (UGC) and VoD [43, 42, 26]. For analytical simplicity, we assume that each content object has the same size which is set to 1. In simulation, the request rate of object n is set to $10^6\lambda(n)$ proportional to the normalized content popularity [requests/time unit].

- **Memory size at the target node:** The total memory size C is set to 1000 and 2000.

- **Default parameters for separate queues in Simplified 2Q /Full 2Q:** In Simplified 2Q, the size of the A_1 queue is set to the number of identifiers for as many objects as would fit on 50% of the total memory size. In Full 2Q, we set the sizes of the A_{1in}, A_{1out}, and A_m queues



(a) $C = 1000$ (b) $C = 2000$

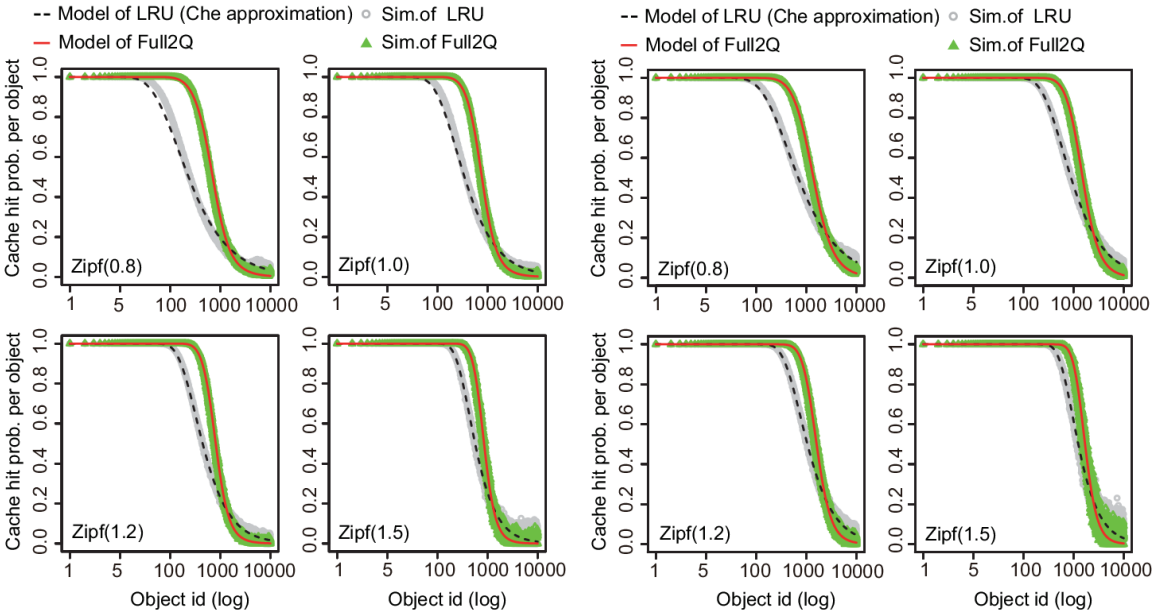
Figure 3.8: Comparison of cache hit probability per object in Simplified 2Q estimated by the proposed model and measured by simulations

to 10, the number of identifiers for as many objects as would fit on 50% of the total memory size, and $C - 10$, respectively.

- **Simulation time :** 60000 time units, e.g. msec.

3.4.1 Approximation Accuracy of Cache Hit Probability in 2Q

We first compare the cache hit probability per object measured by simulations and estimated by the proposed model of Simplified 2Q. Figure 3.8 shows the cache hit probability per object in Simplified 2Q with default parameters. For comparison, we show the cache hit probability per object in LRU estimated by the Che approximation. These results demonstrate that the proposed model of Simplified 2Q can provide a good approximation of the cache hit probability per object measured by simulations. Furthermore, the cache hit probability of warm objects, which are relatively popular but not most popular, are higher than those in LRU. This is why the A_1 queue works well to reduce the influence of *one-timers* on the cache characteristics of warm objects. Moreover as the



(a) $C = 1000$ (b) $C = 2000$

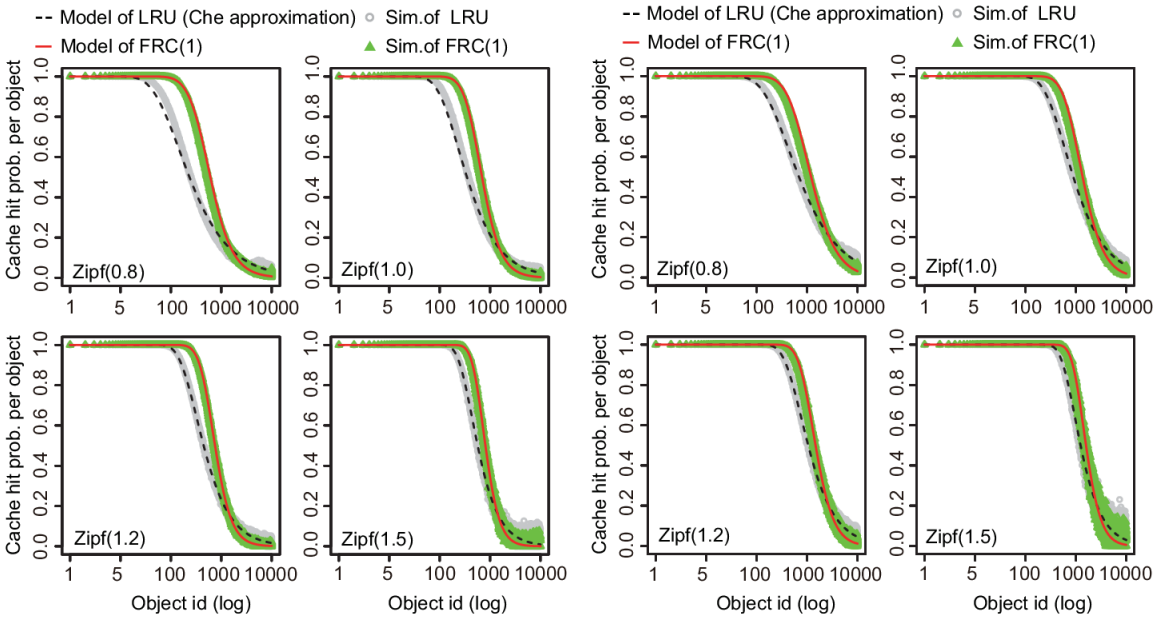
Figure 3.9: Cache hit probability per object in Full 2Q estimated by the proposed model and measured by simulations

Zipf parameter becomes larger, the cache hit probability in Simplified 2Q becomes close to that in LRU.

We further analyze the cache hit probability per object measured by simulations and estimated by the proposed model of Full 2Q with default parameters as shown in Fig. 3.9. As with the results of Simplified 2Q, the proposed model of Full 2Q can provide a good approximation of the cache hit probability per object measured by simulations. We can see that these results in Full 2Q are almost same as those in Simplified 2Q.

3.4.2 Approximation Accuracy of Cache Hit Probability in FRC and ARC

In this section, we demonstrate that the proposed model of FRC(1) can approximate the cache performance of ARC. In Fig. 3.10, we first present the cache hit probability estimated by the proposed model of FRC(1) and measured by simulation. These results show that the proposed model can approximate the simulation results and can reduce the influence of *one-timers* on the cache hit



(a) $C = 1000$ (b) $C = 2000$

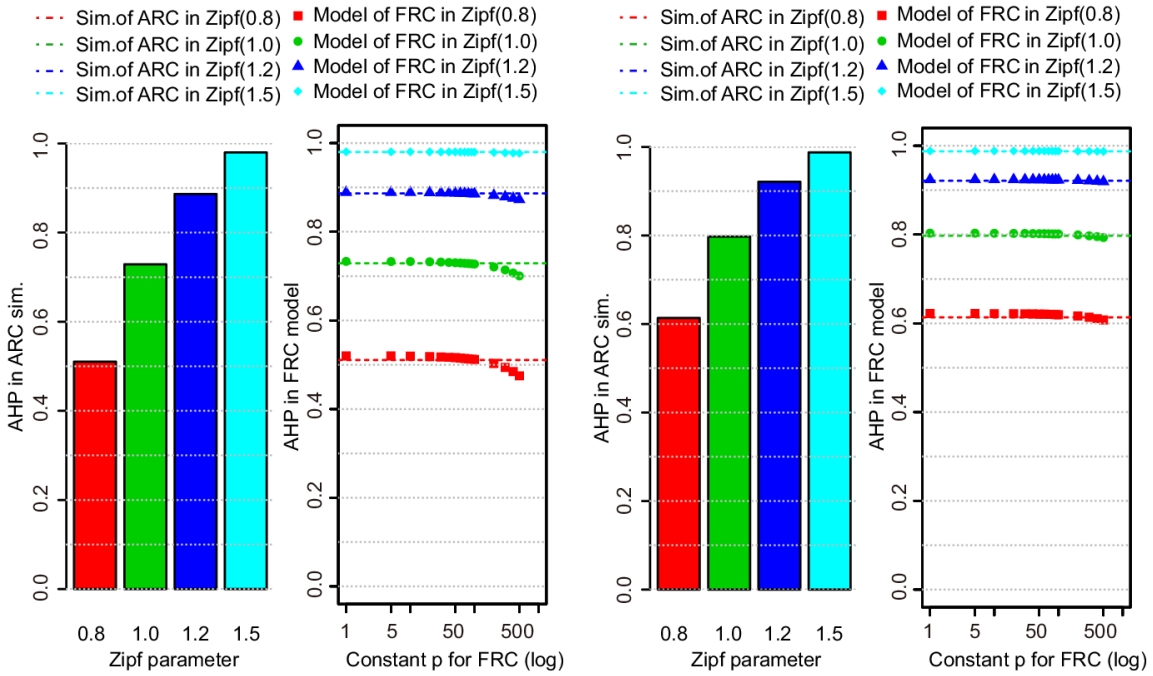
Figure 3.10: Cache hit probability per object in FRC(1) estimated by the proposed model and measured by simulations

probability of warm objects by using the L_1 list in FRC.

Next we evaluate the average hit probability of all objects estimated by the FRC model for different constant p and measured by simulations of ARC in Fig. 3.11. Here, we define the average hit probability (AHP) in the approximation models and simulations as follows.

- Model: $\mathrm{AHP}_{model} = \lambda(n)H(n)$, where $H(n)$ is the cache hit probability of object n estimated by the model.
- Simulation: $\mathrm{AHP}_{sim} = \frac{\text{Total number of hits of all objects}}{\text{Total number of requests for all objects}}$

These results demonstrate that the average hit probability in FRC with small p are almost the same regardless of p and can approximate those measured by simulations of ARC. As a reference, we show the average values of p in simulations of ARC which are adaptively tuned according to nonstationary accesses in Fig. 3.12. In all conditions, the values of p are controlled at a low level. These results suggest that the size of the T_1 queue should be small and the B_1 queue should be large



(a) $C = 1000$ (b) $C = 2000$

Figure 3.11: Average hit probability of all objects estimated by the FRC model when changing p and measured by simulations of ARC

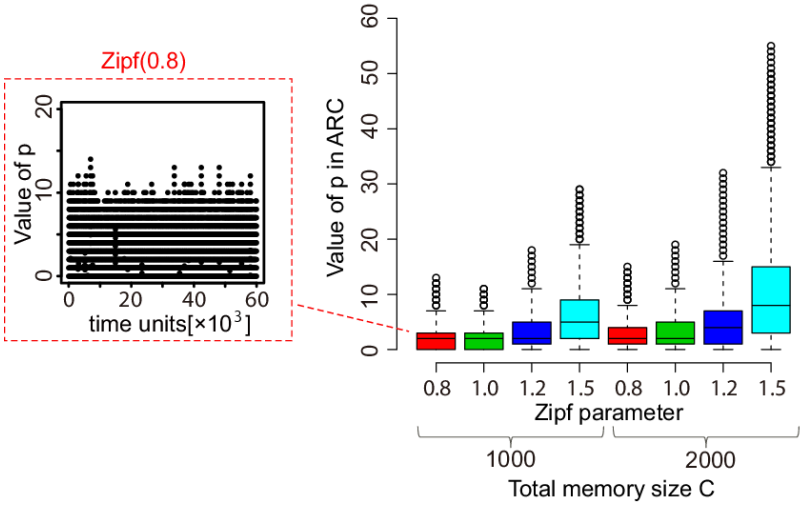
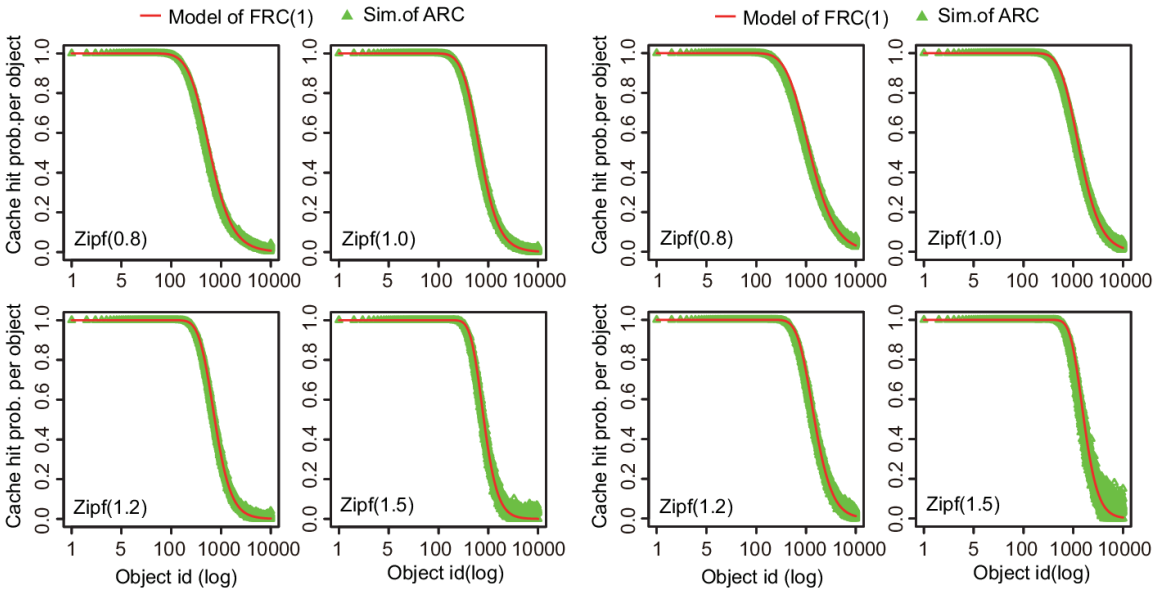


Figure 3.12: Box plot of values of p in simulations of ARC ($C = 1000$ and 2000)



(a) $C = 1000$ (b) $C = 2000$

Figure 3.13: Cache hit probability per object estimated by the proposed model of FRC(1) and measured by simulations of ARC

to improve the cache performance at steady state. Since *one-timers* are actually rare events, it is not necessary to cache the data of *one-timers* and sufficient to only store their pointers. As a result, we can see that FRC(1) is suitable as an approximation model of ARC. Additionally, Fig. 3.13 presents the cache hit probability per object estimated by the proposed model of FRC(1) and measured by simulations of ARC. As a result, we see that the proposed model of FRC(1) can provide a highly accurate approximation of the cache hit probability per object in ARC.

3.4.3 Comparison of Performance in Each Caching Mechanism

Next, by using the approximation models, we evaluate the cache performance for Simplified 2Q, Full 2Q, ARC, and LRU.

Figure 3.14 presents the average hit probability AHP estimated by the proposed models of 2Q and ARC, as well as the Che approximation model for LRU. The AHP becomes higher as the Zipf parameter becomes larger. Furthermore, we can see that the AHP in Simplified 2Q, Full

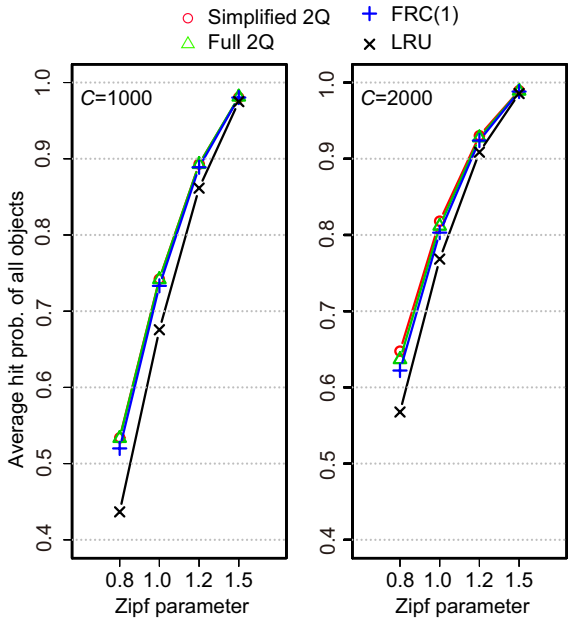


Figure 3.14: Cache hit probability of all objects estimated by each approximation model

2Q, and ARC are almost the same and better than those of LRU. This is why the 2Q and ARC mechanisms can mitigate the cache pollution of *one-timers* and the cache hit probability of warm objects improves the average cache hit probability of all objects when the Zipf parameter is small as shown in Figs. 3.8, 3.9, and 3.10. On the other hand, we confirmed that the proposed models can estimate the cache performance at steady state for 2Q and ARC in a short time compared with the simulation-based evaluations. In this way, the proposed models can easily evaluate the cache characterisics of 2Q and ARC, which are influenced by node memory size and access patterns, and we can use the proposed models as design guidelines for 2Q and ARC cache systems.

3.5 Summary

We proposed approximation models of 2Q and ARC which can consider the interactions between separate queues for *one-timers*. In the evaluations, we validated that the proposed models can provide a highly accurate approximation of the simulation results. As a result, we showed that the proposed models can easily analyze the statistical performance of 2Q and ARC at high accuracy. While

the simulation-based evaluations actually take a long time to measure the cache characteristics at steady state, the proposed models can estimate the statistical performance for various conditions in a short time. Finally, by using the model-based analysis, we confirmed that the cache performance of 2Q and ARC is better than that of LRU because the partition management of multiple queues can reduce the influence of *one-timers* on the cache performance of relatively popular objects.

In this research, we didn't discuss the influence of the 2Q and ARC mechanisms of a cache node on the whole performance of the caching network but can easily enhance the proposed models by applying the search method of characteristic times for caching networks in [44].

Chapter 4

Modeling and Evaluation of Static TTL Management in Hierarchical Caching

4.1 Issue and Approach for Analysis of Static TTL Management

In-network caching technologies are expected to reduce the network traffic and improve the service quality, such as communication latency, by storing content data on network nodes near to users. Meanwhile, the adaptive cache management using TTL of content can realize efficient memory management per content. In a distributed cache system like CCN, it is an important issue to evaluate cache performance and network resources required in the cache mechanism using the TTL value.

In TTL-based caching, each *CN* resets the time counter to the TTL of content every time a new request for this content arrives and decreases the counter by 1 every time unit (cf. Fig. 4.1). In this mechanism, each *CN* caches data of content delivered by another *CN* or an origin server when the content counter is above 0 and discards the data of content when the counter becomes 0. Meanwhile in ICN, each *CN* autonomously constructs some caching hierarchies rooted at each origin site of content (cf. Fig. 1.1). The caching hierarchy is constructed by routes between the origin site, caching nodes and users, such that less popular content is cached on *CN*s near to the origin site and more popular content is cached on *CN*s near to users. Therefore, it is difficult to evaluate the impact of TTL-based caching on network resources and performance because the characteristics in

the distributed cache mechanism depend on the caching hierarchies with distributed cache nodes.

As a enhanced model of LRU and FIFO policies, Fofack *et al.* [48, 49] introduce a TTL-based caching model. Furthermore, Carofiglio *et al.* [51] explore the impact of storage management on the cache performance per application in CCN and evaluate the effectiveness of static storage partitioning and dynamic management by priority-based weighted fair schemes combined with TTL-based caching. Moreover, they study the possibility of improving cache scalability in TTL-based caching without cache coordination. Hou *et al.* [52] propose an analytical model of a hierarchical TTL-based caching system. The proposed mechanism updates content objects by using the TTL values which are randomly set for each node according to its layer on the delivery tree.

However, these proposals don't discuss the cache characteristics using TTL of content and the impact of TTL on network resources and cache performance on the multiplexed caching hierarchies. Therefore, we investigate a statistical model to analyze the influence of TTL for content, on hierarchical caching systems. This study assumes that each *CN* executes data caching using TTL of content which can, for instance, be signaled in the data header of the content or set at each *CN* in advance.

In this chapter, to provide a design guideline for caching networks using TTL and analyze the cache characteristics in the steady state, we first propose an analytical model using matrix equations to evaluate the cache characteristics on multiplexed caching hierarchies of content and evaluate the validity of the proposed model and the impact of the TTL value.

The remainder of this chapter is organized as follows. We propose our analytical model in Section 4.2 and demonstrate evaluation results using the proposed model in Section 4.3. Furthermore, in Section 4.4 we introduce a caching mechanism to improve energy efficiency using TTL and we evaluate the effectiveness of the proposed mechanism. Finally, we summarize this chapter in Section 4.5.

4.2 Analytical Model

We first propose the evaluation model to analyze the cache performance using TTL of a content in the distributed cache system having multiplexed caching hierarchies.

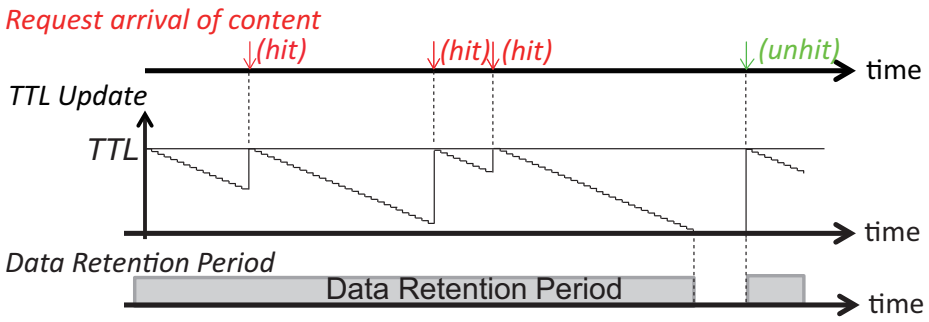
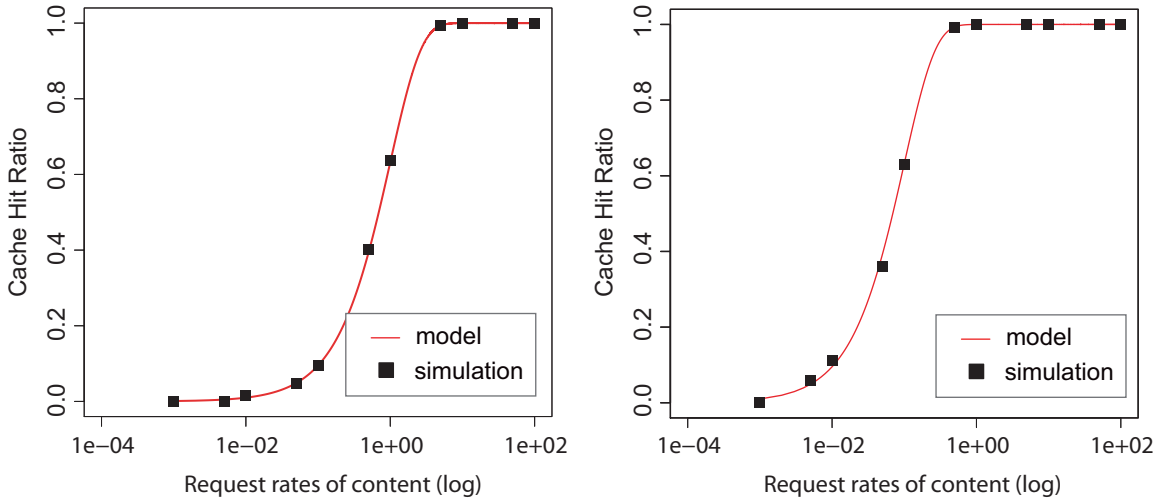


Figure 4.1: Traditional TTL-based caching



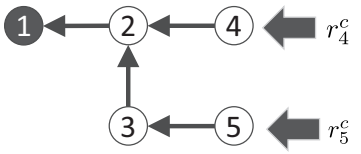
(a) $TTL = 1$ (b) $TTL = 10$

Figure 4.2: Cache hit ratio when the requests per content with the rate λ input to a *CN* at an exponentially distributed interval and set "total request rates of a content item [requests/sec]" and "TTL [sec]" to various patterns

In TTL-based caching, the cache probability of content c having request rates λ^c to a *CN* can be expressed by the following function [34].

$$f(\lambda^c, TTL^c) = 1 - e^{-\lambda^c TTL^c}$$

As shown in Fig. 4.2, we demonstrate that this statistical function can provide a good approximation of the cache hit ratio of content at a *CN* under the assumption that content requests arrive as a



$$\mathbf{\Lambda}^c[s] = \begin{bmatrix} \lambda^c_{(1,4)}[s] & \lambda^c_{(1,5)}[s] \\ \lambda^c_{(2,4)}[s] & \lambda^c_{(2,5)}[s] \\ \lambda^c_{(3,4)}[s] & \lambda^c_{(3,5)}[s] \\ \lambda^c_{(4,4)}[s] & \lambda^c_{(4,5)}[s] \\ \lambda^c_{(5,4)}[s] & \lambda^c_{(5,5)}[s] \end{bmatrix} \quad \mathrm{R}^c = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ r_4^c & 0 \\ 0 & r_5^c \end{bmatrix}$$

$$\mathrm{D}^c[s] = \begin{bmatrix} 0 & e^{-\sum_{j=4}^{5} \lambda^c_{(2,j)} TTL^c} & 0 & 0 & 0 \\ 0 & 0 & e^{-\sum_{j=4}^{5} \lambda^c_{(3,j)} TTL^c} & e^{-\sum_{j=4}^{5} \lambda^c_{(4,j)} TTL^c} & 0 \\ 0 & 0 & 0 & 0 & e^{-\sum_{j=4}^{5} \lambda^c_{(5,j)} TTL^c} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.3: An example of matrices $\mathbf{\Lambda}^c$, $\mathbf{R}^c$, and D^c for the request propagation on the delivery tree having origin 1

Poisson process. We next show the matrix model of request propagation of content in TTL-based caching on caching hierarchy of content. The propagation of each request (*Interest*) of content c on its caching hierarchy is expressed by the following model.

$$\mathbf{\Lambda}^c[s+1] = \mathrm{D}^c[s] \cdot \mathbf{\Lambda}^c[s] + \mathbf{R}^c, \forall c \tag{4.1}$$

Here, M, N, and s are the number of *CN*s, the number of sites having requesting users, and the number of steps that each request propagates to the next *CN*, respectively. Moreover, we define $\mathbf{\Lambda}^c$ as the $M \times N$ matrix consisting of the request rates $\lambda^c_{(i,j)}$ of content c from the requesting user in site j to CN_i and $\mathbf{R^c}$ as the $M \times N$ matrix of which elements are the request rates r_i^c of content c from users in site i.

$$\mathbf{\Lambda}^c[s] := [\lambda^c_{(i,j)}]_{M \times N}$$

$$[\mathbf{R}^c]_{ij} := \begin{cases} r_j^c, \ \forall i = req\_site(j) \\ 0 \quad \text{otherwise} \end{cases}$$

D^c is the $M \times M$ matrix of request propagation for content c as follows.

$$[\mathrm{D}^{\mathrm{c}}]_{mn} := \begin{cases} 1 - f(\sum_k^N \lambda_{(n,k)}^c[s], TTL^c), & \forall m = parent\_node(n) \\ 0 \quad \text{otherwise} \end{cases} \tag{4.2}$$

In Fig. 4.3, we show an example of these matrices for the delivery tree having origin 1.

In the iterative matrix equation, we can consider the request propagation process and data caching at each *CN* for content requested from each site. Moreover, the steady state of network resources and cache performance per content can be derived by iteratively calculating the equation s_{max}-times which is the maximum number of hops from each site having requesting users to its origin site.

Using the proposed model, we can model the system state and the cache performance for content c such as

- memory usage per content in each *CN*,
- the total amount of transmission data in the network
- power consumption which is the sum of "cache allocation power" and "traffic transmission power".
- cache hit ratio per content which is the probability that the content is cached in the networks, and
- average hop length per content.

4.2.1 Memory Usage

The memory usage of content c at CN_i is derived using the data size θ_c of content c as follows.

$$\mathrm{U}_i^c := \theta_c f(\sum_k^N \lambda_{(i,k)}^c, TTL^c). \tag{4.3}$$

4.2.2 Transmission Data

The total amount of data delivering of content c through all *CN*s is derived by

$$\mathrm{Dt}^c := \theta_c \sum_j^N Tr_j^c \tag{4.4}$$

using the following vector consisting of the cumulative number of traffic flows Tr_j^c through each *CN* on the delivery route for content c having origin site o requested by users in site j.

$$\begin{aligned}\mathbf{Tr}^c &:= [Tr_1^c \cdots Tr_j^c \cdots Tr_N^c]^{\mathrm{T}} \\ &= (\mathbf{H} * \mathbf{\Lambda}^c)^{\mathrm{T}} \begin{bmatrix} f(\sum_k^N \lambda_{(1,k)}^c, TTL^c) \\ \vdots \\ f(\sum_k^N \lambda_{(M,k)}^c, TTL^c) \end{bmatrix} + (\mathbf{H}[o,] * \mathbf{\Lambda}^c[o,])^{\mathrm{T}} (1 - f(\sum_k^N \lambda_{(o,k)}^c, TTL^c))\end{aligned} \tag{4.5}$$

Here, we define "$*$" as the element-wise product of a matrix or vector and $\mathbf{H} = [h_{(i,j)}]_{M \times N}$ as the matrix consisting of shortest hop length $h_{(i,j)}$ from CN_i to CN_j.

Moreover, the second term in Eq. (4.5) presents the amount of transmission data which aren't cached on the network.

4.2.3 Power Consumption

we consider total power consumption based on *Energy Proportional Networks* [21, 22] in which power consumption of each device is proportional to its usage. In this chapter, we assume 1 time unit as 1 sec.

Cache allocation power: CP^c [J] for storing data of content c in 1 sec, i.e., the total power consumed by storing content c on each *CN* in the network, is defined as

$$\mathrm{CP}^c := \theta_c P_{ca} \sum_i^M f(\sum_k^N \lambda_{(i,k)}^c, TTL^c), \tag{4.6}$$

where P_{ca} is the memory power density [J/(bit· s)].

Traffic transmission power: TP^c [J] i.e., the total power consumed by network devices when

data of content c are delivered on the shortest routes, is derived as

$$\mathrm{TP}^c := (P_r + P_{wdm})\mathrm{Dt}^c, \tag{4.7}$$

where P_r and P_{wdm} are the power densities [J/bit] of a router and of a WDM node along the delivery routes, respectively.

4.2.4 Cache Hit Ratio

The cache hit ratio of content c having origin o in the network is derived as

$$\mathrm{CHR}^c := 1 - \frac{\sum_j^N \lambda_{(o,j)}^c \left(1 - f(\sum_k^N \lambda_{(o,k)}^c, TTL^c)\right)}{\sum_j^N r_j^c}. \tag{4.8}$$

4.2.5 Average Hop Length

The average hop length of content c having origin o is derived as

$$\mathrm{AHL}^c := \frac{\sum_j^N \left(Tr_j^c + \mathrm{Hp}^o \lambda_{(o,j)}^c (1 - f(\sum_k^N \lambda_{(o,k)}^c, TTL^c)\right)}{\sum_j^N r_j^c}. \tag{4.9}$$

Where, the second term of the numerator is a penalty for the hop length of content c which isn't cached on any *CN*s in the network and for which a request reaches its origin server and Hp^o is the hop-length from the origin server to the *CN* in origin site o.

All variables in the proposed model are summarized in Table 4.1.

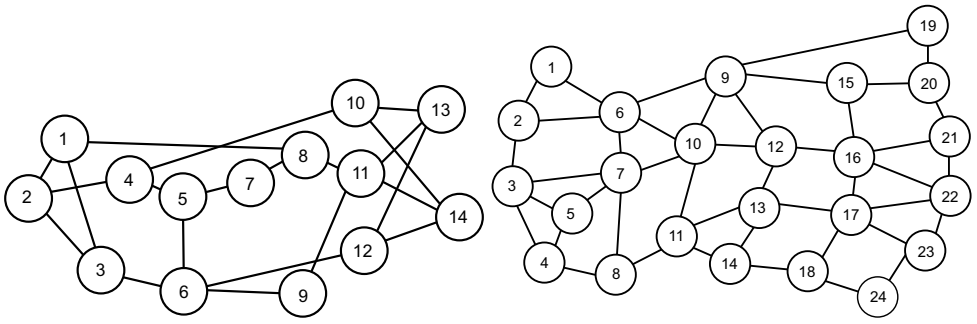
4.3 Evaluation using the Proposed Model

We evaluate the cache characteristics in TTL-based caching when changing the TTL value of content. The evaluation conditions are set to the following.

- **Test networks:** NSF topology with 14 *CN*s (Topology A), cf. Fig. 4.4(a) / US-backbone topology with 24 *CN*s (Topology B), cf. Fig. 4.4(b). The maximum number of hops (s_{max}) is

Table 4.1: Variables in the proposed model

Variable	Definition
M	The number of *CN*s
N	The number of sites having requesting users
θ_c	Data size of content c
$\lambda^c_{(i,j)}$	Request rates to CN_i for content c requested by users in site j
r^c_i	Request rate of content c requested by users in site i
TTL^c	TTL of content c at CN_i
U^c_i	Memory usage of content c at CN_i
Dt^c	Total amount of data delivery of content c through all *CN*s
Tr^c_j	Cumulative number of traffic flows through each *CN* on the delivery route for content c requested by users in site j
CHR^c	Cache hit ratio of content c in the network
AHL^c	Average hop length of content c
$h_{(i,j)}$	Shortest hop length from CN_i to CN_j
Hp^o	Hop length from origin server to the *CN* in origin site o
CP^c	Total power consumption [J] for data storage of content c in 1 sec
TP^c	Total power consumption [J] delivering content c on the delivery routes
P_{ca}	Power density for storage [J/(bit·s)]
P_r	Power density of a router [J/bit]
P_{wdm}	Power density of a WDM node [J/bit]



(a) Test topology A (b) Test topology B

5 hops in Topology A and 7 hops in Topology B. Furthermore, we assume that the memory size of each *CN* is infinite and each site has requesting users for all content items, which means M is equal to N. For evaluation, we set Hp^o to 5 as the penalty of hop length.

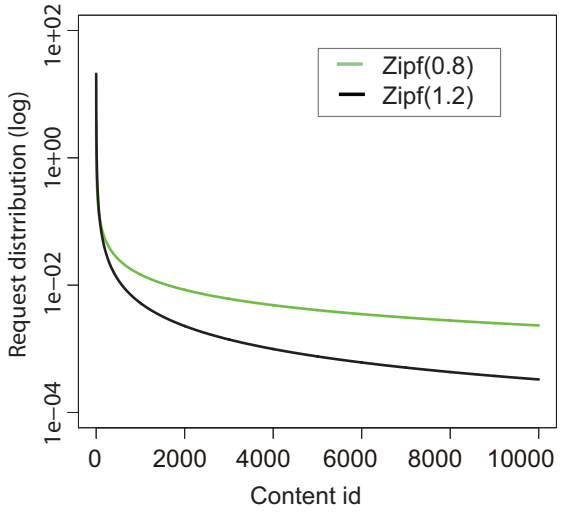


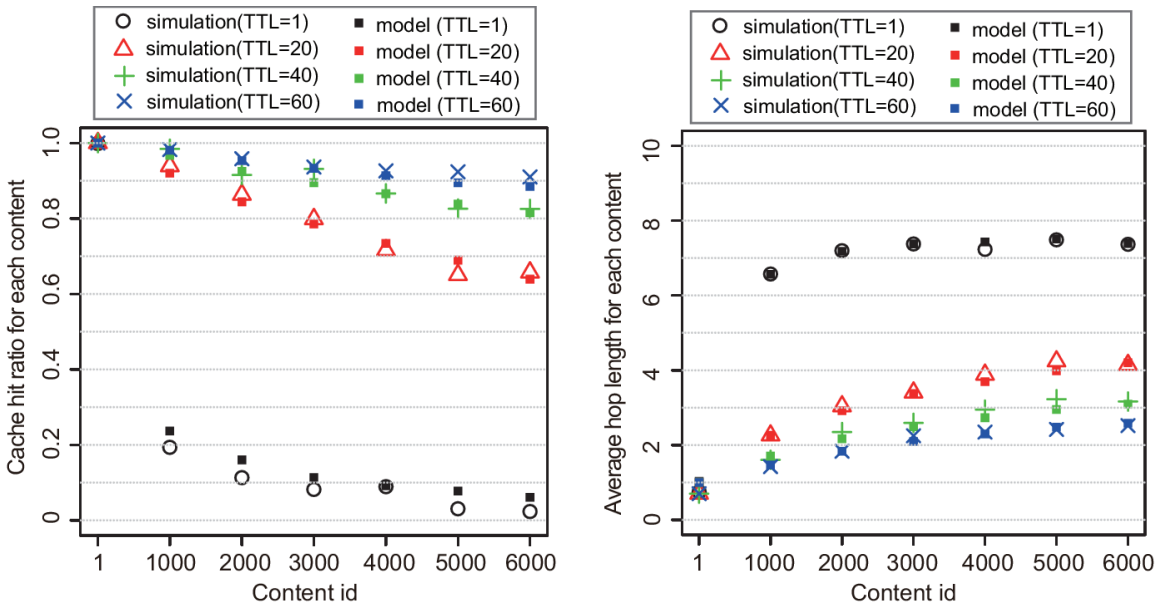
Figure 4.4: Evaluation conditions r_j^c

- **Content information:** Zipf-distributed requests from each site j for $K = 10000$ content items are defined as $r_j^c = \gamma k^{-\alpha}/c$, $c = \sum_{k=1}^{K} k^{-\alpha}$, cf. Fig. 4.3. We set α to 0.8 for UGC (User Generated Content) and 1.2 for VoD [42] and γ to 100 [requests/sec]. Furthermore, the origin site t of content ID k is set randomly based on a uniform distribution. The content size is geometrically distributed with mean 10 MB [43].

4.3.1 Verification of the Proposed Model

To verify the proposed model, we compare the cache performance using the model with that measured by simulations for 7 content items with $\alpha = 0.8$ in Topology A. In the evaluations, we set the TTL value as $\{1, 20, 40, 60\}$ [sec].

Figures 4.5(a) and 4.5(b) shows that the cache hit ratio and average hop length for each content provide suitable approximations of the simulation results. As a result, we see that the proposed model can express the statistical characteristics for TTL-based caching.



(a) Cache hit ratio

(b) Average hop length

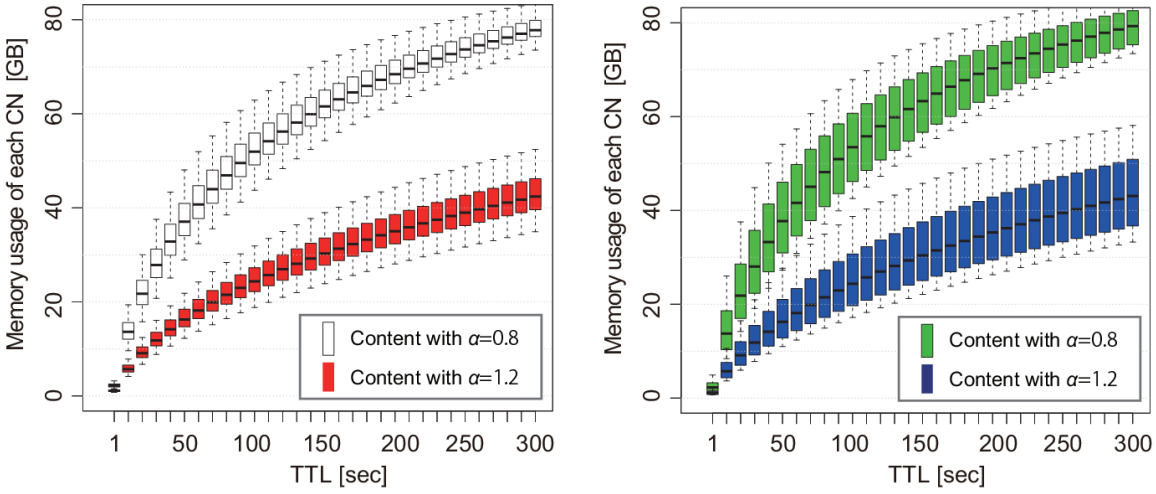
Figure 4.5: Cache performance estimated by the proposed model and calculated by simulations for different content ids

4.3.2 Impact of TTL

For the next evaluation, we define TTL of all content as the same value which is changed from 1 [sec] to 300 [sec] on the assumption that the TTL value is signaled in the data header of content.

Figure 4.6 shows the memory usage at each *CN* when the TTL value is changed. In these results, the memory usage of each *CN* becomes larger as the TTL value becomes larger. Moreover, the memory usage for content with $\alpha = 0.8$ is larger than that for content with $\alpha = 1.2$ because less popular content with $\alpha = 0.8$ has higher request rates and is easier to be cached than that with $\alpha = 1.2$.

Furthermore, Fig. 4.7 shows the power consumption of the target network according to the change of the TTL value using the power densities of network devices shown in Table 2.3 of Chapter 2. In addition, Fig. 4.8 presents cache hit ratio for all content items calculated by Eq. (4.8) and average hop length for all content items calculated by Eq. (4.9).



(a) Topology A

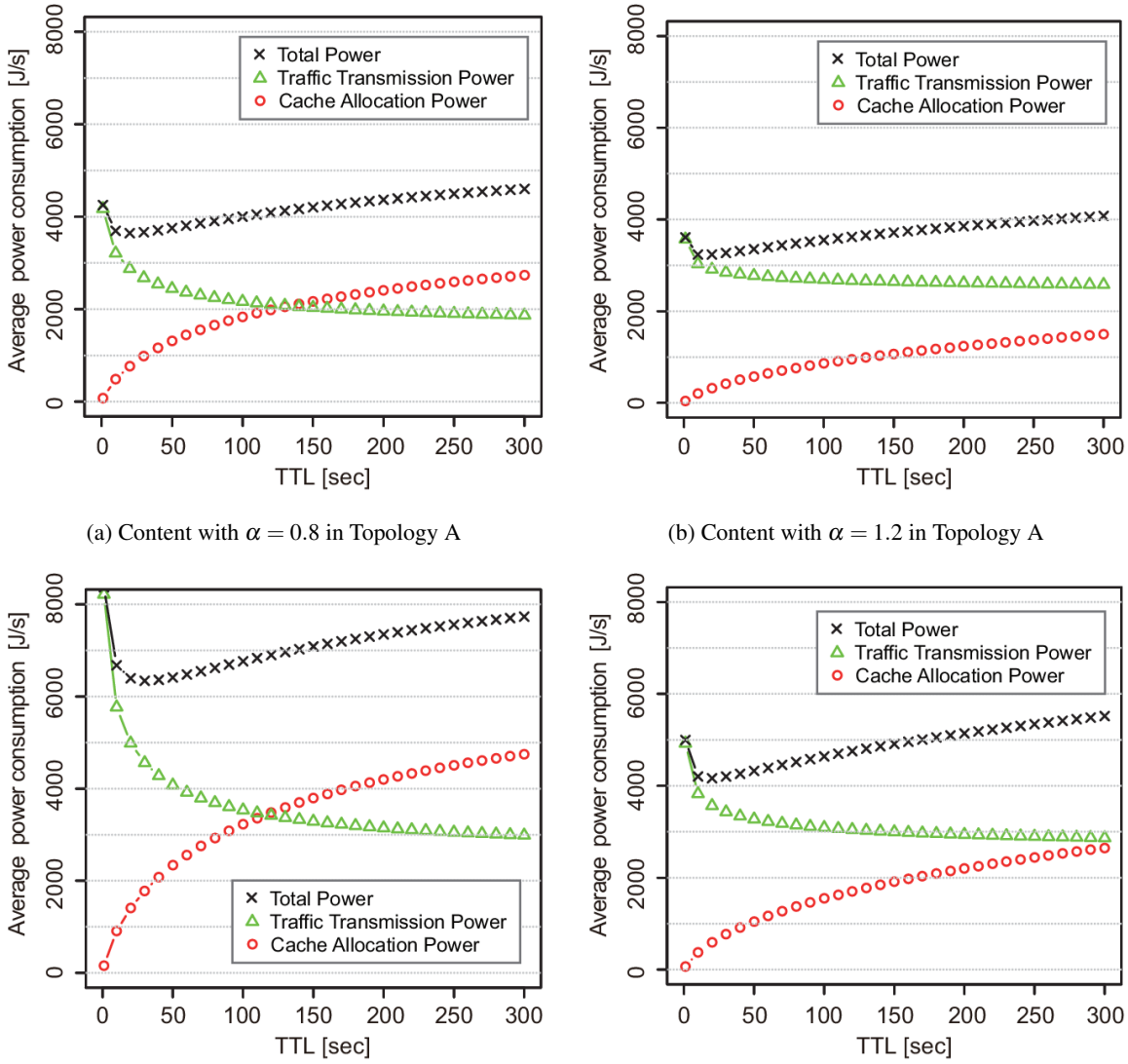
(b) Topology B

Figure 4.6: Box plot of memory usage at each *CN* when the TTL value is changed

Figure 4.7 demonstrates the tradeoff between *cache allocation power* and *traffic transmission power* for the change of the TTL value. Figs. 4.7(a) and (c) show that there is a point reversing the relation of *cache allocation power* and *traffic transmission power* for the TTL value. Therefore, the energy impact of TTL is also different depending on the network conditions and the proposed model can search for the energy efficient TTL in consideration of the tradeoff of power consumption.

Meanwhile in Fig. 4.8(a), the cache hit ratio is also low in the region of the TTL values leading to lower power consumption. Therefore, we should consider the relation between cache hit ratio and power consumption to search for the energy efficient TTL. Furthermore, Fig. 4.8(b) shows that the average hop length of all content items becomes smaller as the TTL becomes larger. In these results, the approaching of the average hop length 1 hop means that all content items are cached in all *CN*s. Therefore, the cache hit ratio approaches to around 100 % as the average hop length is approaching to 1 and the memory usage becomes larger.

As a result, using the proposed model, we can analyze the cache characteristics in the distributed cache system and provide a design guideline for TTL of content in view of energy efficiency or efficient memory usage in each *CN*. Next we introduce an energy efficient caching mechanism using TTL as application and demonstrate the effectiveness of the energy efficient TTL.



(a) Content with $\alpha = 0.8$ in Topology A

(b) Content with $\alpha = 1.2$ in Topology A

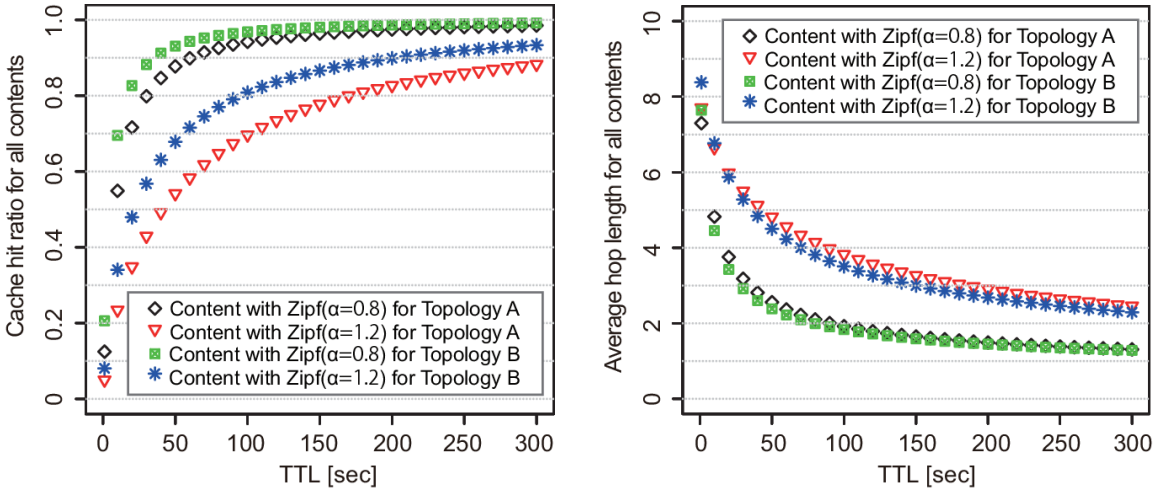
(c) Content with $\alpha = 0.8$ in Topology B

(d) Content with $\alpha = 1.2$ in Topology B

Figure 4.7: Power consumption of the network when the TTL value is changed

4.4 Application to a Caching Mechanism using Energy Efficient TTLs

In consideration of energy efficiency in content dissemination networks, we previously proposed an ILP model to design the most energy efficient cache locations in consideration of the multiplexed caching hierarchies [16].



(a) Cache hit ratio for all content items

(b) Average hop length for all content items

Figure 4.8: Cache performance when the TTL value is changed

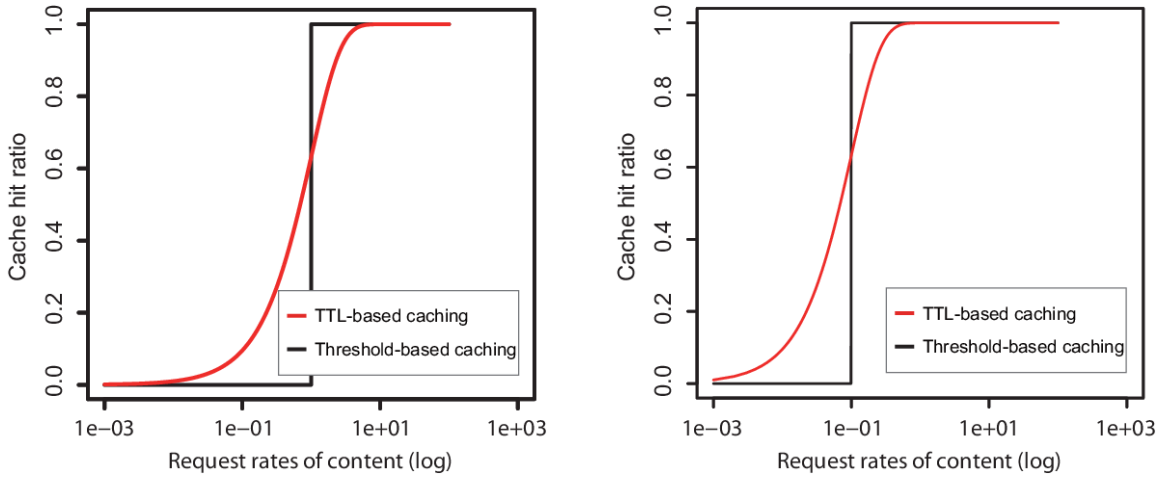
In [17], we proposed the threshold-based caching mechanism to locally search for locations which are near to the most energy efficient locations. In threshold-based caching, every *CN* automatically pre-designs a threshold of request rates of content using local information on each caching hierarchy before cache operation and the content data are cached when the request rate of the content is above a pre-designed threshold or isn't cached when the request rate is below that threshold.

In threshold-based caching, CN_i has the threshold Th_i^o [requests/sec] of request rates for origin site o of content, which is uniquely determined for each delivery tree in the target network. Furthermore, we can express the request propagation matrix D_{th}^{c} of threshold-based caching in the proposed model as

$$[\mathrm{D}_{th}^{c\in \mathrm{C_o}}]_{mn} := \begin{cases} 1 & \forall m = parent\_node(n) \wedge \sum_k^N \lambda_{(n,k)}^c[s] < Th_n^o \\ 0 & \text{otherwise} \end{cases} . \tag{4.10}$$

$\mathrm{C_o}$ is the set of content items having origin o.

In this chapter, we propose an approximation method using TTL of threshold-based caching because TTL-based caching can realize a more simple cache management by just updating the TTL



(a) $TTL = 1, Th = 1$ (b) $TTL = 10, Th = 0.1$

Figure 4.9: Comparison of cache hit ratio at a *CN* with threshold-based caching and TTL-based caching

counter of content without having to measure the request rates of content like in threshold-based caching.

We derive the approximation method using TTL of threshold-based caching as follows.

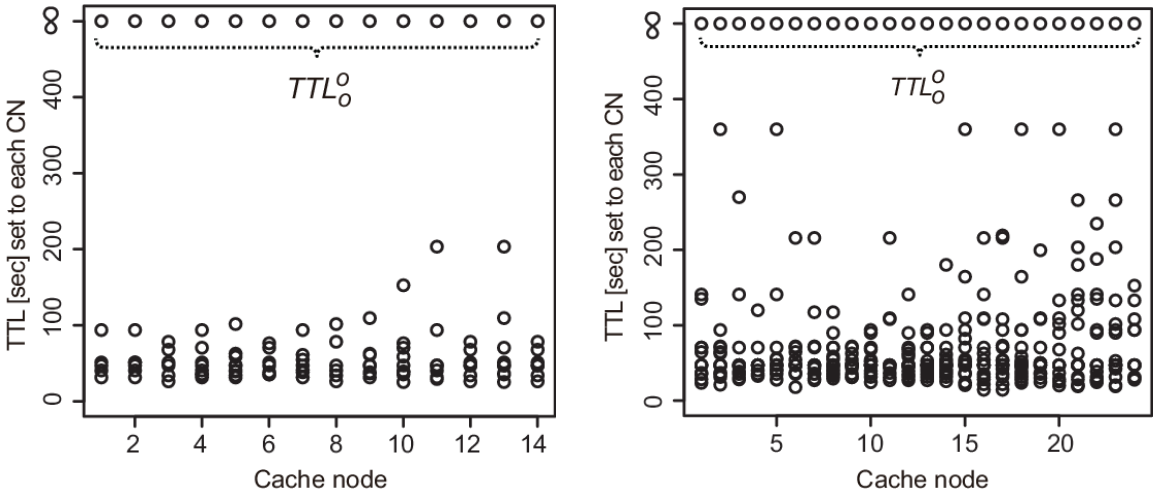
$$TTL_i^o = \frac{1}{Th_i^o}, \forall i, o \tag{4.11}$$

Here, TTL_i^o is set to CN_i and defined as a different value for each origin site o of content.

In Fig. 4.9, we evaluate the cache hit ratio of content at a *CN* for threshold-based caching and the energy efficient TTL-based caching. As a result, we see that the cache hit ratio in the energy efficient TTL can provide similar characteristics to that in threshold-based caching.

Using Eq. (4.1), we can derive the request propagation matrix $\mathrm{D}_{ttl}^{c\in \mathrm{C_o}}$ using the energy efficient TTL as

$$[\mathrm{D}_{ttl}^{c\in \mathrm{C_o}}]_{mn} := \begin{cases} 1 - f(\sum_k^N \lambda_{(n,k)}^c[s], \frac{1}{Th_n^o}), & m = parent\_node(n) \\ 0 \quad \text{otherwise} \end{cases}. \tag{4.12}$$



(a) Topology A

(b) Topology B

Figure 4.10: Energy efficient TTL for each topology

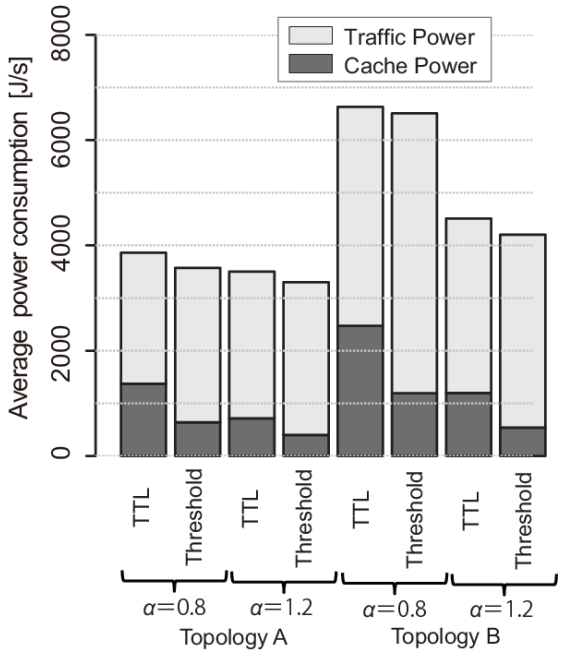


Figure 4.11: Power consumption for energy efficient TTL-based caching and threshold-based caching

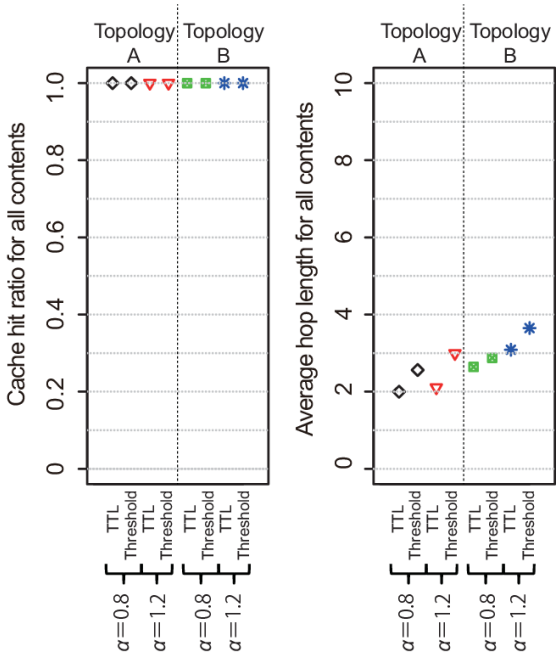


Figure 4.12: Cache performance for energy efficient TTL-based caching and threshold-based caching

In these caching mechanisms, the threshold Th_o^o and the TTL $\frac{1}{Th_o^o}$ for content having origin o in CN_o are defined as 0 and ∞, respectively. Therefore, all content items are always cached in the network unless memory overflow occurres in each CN. Here, we demonstrate the effectiveness of the energy efficient TTL based on the same conditions in Section 4.3. Fig. 4.10 shows the TTL values derived by Eq. (4.11). In these results, TTL_o^o at CN_o in origin site o is infinite and the other TTLs are derived as different values for each target network.

In Figs. 4.11 and 4.12, we compare the total power consumption and the cache performance for two mechanisms using the energy efficient TTLs and thresholds of request rates, respectively. In these results, the total power consumption in the energy efficient TTL-based caching is near to that in threshold-based caching. Moreover, the cache hit ratio is always 100% because TTL_o^o and Th_o^o $(\forall o)$ are infinite and 0. The average hop length in TTL-based caching is slightly smaller than that in threshold-based caching because the memory usage in TTL-based caching is larger than that in threshold-based caching as shown in Fig. 4.13.

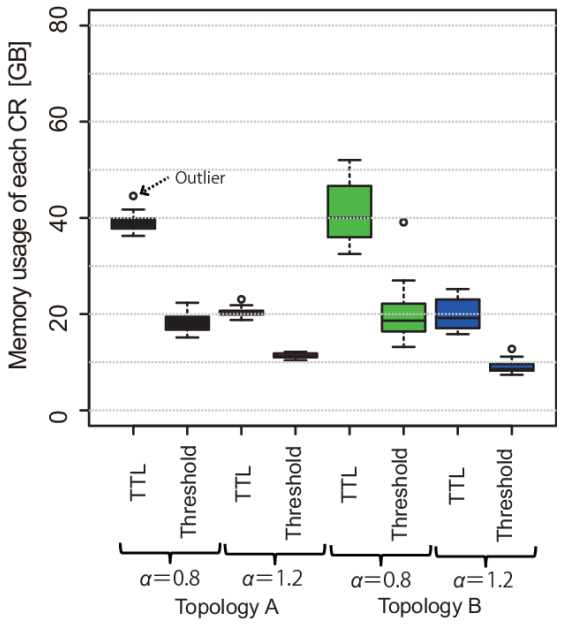


Figure 4.13: Box plot of memory usage for TTL-based caching and threshold-based caching

4.5 Summary

We proposed an analytical model to evaluate the cache characteristics of a distributed cache system like CCN. The proposed model is expressed by iterative matrix equations and can evaluate the impact of TTL-based caching on network resources and cache performance on multiplexed caching hierarchies. In the evaluations, we verified the varidity of the proposed model and analyzed the impact on memory usage, power consumption, cache hit ratio, and average hop length when changing the TTL value of content.

Furthermore, we introduced the energy efficient TTL to reduce the power consumption of the network and evaluated its effectiveness. Based on the proposed model, we showed that the energy efficient TTL-based caching can achieve a similar power consumption like threshold-based caching that searches for the most energy efficient cache locations.

Chapter 5

Design, Modeling, and Evaluation of Adaptive TTL Management in Hierarchical Caching

5.1 Issue and Approach for Adaptive TTL Management

The cache aging techniques based on TTL of content facilitate analyzing cache characteristics and can realize appropriate resource management by setting efficient TTLs. Therefore, it becomes an important issue to manage system resources, such as storage and network bandwidth, which are influenced by the TTL value of content.

Traditionally, there are content placement algorithms [12, 53, 54, 55] as a solution for *File Allocation Problems* [56] which minimize the cost imposed for content storage and queries (requests), or maximize the performance such as distance to content. Baev *et al.* [53] propose an Integer Linear Programming (ILP) model which minimizes content placement cost and an approximation solution using a linear relaxation. Furthermore, Qui *et al.* [54] propose some replica placement algorithms to solve the K-median problem for CDNs.

In contrast to the above-mentioned content placement problems, Borst *et al.* [12] formulate an ILP model based on a hierarchical structure for content locations to minimize bandwidth costs and

propose a distributed solution of the problem. Moreover, they evaluate the cost-saving effect for a hierarchical topology which has symmetric bandwidth cost for a parent node and some leaf nodes.

Fofack *et al.* [48, 49] introduce a TTL-based caching model. Moreover, they [49] propose iterative calculation methods for the approximation models, which can analyze the performance in TTL-based cache networks, and demonstrate that the proposed models can approximate cache hit ratio for some caching networks at high accuracy. Additionally, Berger *et al.* [35] also model cache characteristics of two TTL-based caching, and the combination of the both. In Chapter 4, we proposed an analytical model using simple matrix equations, which can analyze cache performance and request propagation from all requesting users.

However, these proposals don't discuss adaptive resource control to reduce the resource cost in hierarchical caching by searching for an appropriate TTL of content. Therefore, as an enhanced approach of TTL-based caching, we investigate a control methodology to efficiently use system resources consisting of *storage cost* and *bandwidth cost* by adaptively tuning the TTL value of each content object.

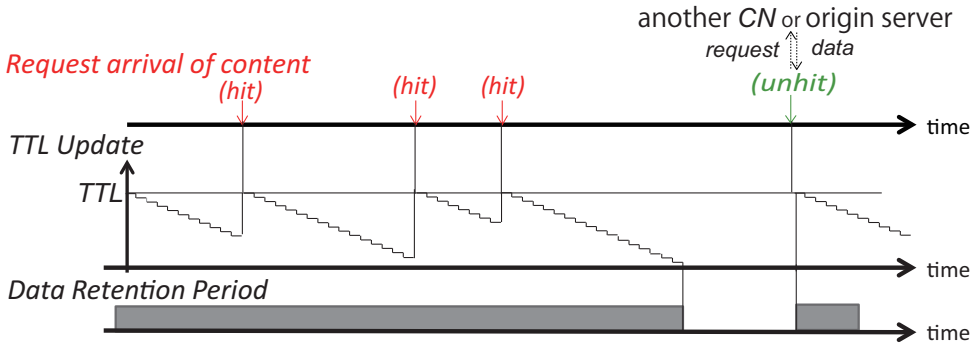
In this chapter, we assume the following policies in TTL-based caching [11, 34, 35].

- *Policy 1:* Each *CN* resets the time counter to the TTL value for each content every time a new request for this content arrives and decreases the counter by 1 every time unit (cf. Fig. 5.1(a)).

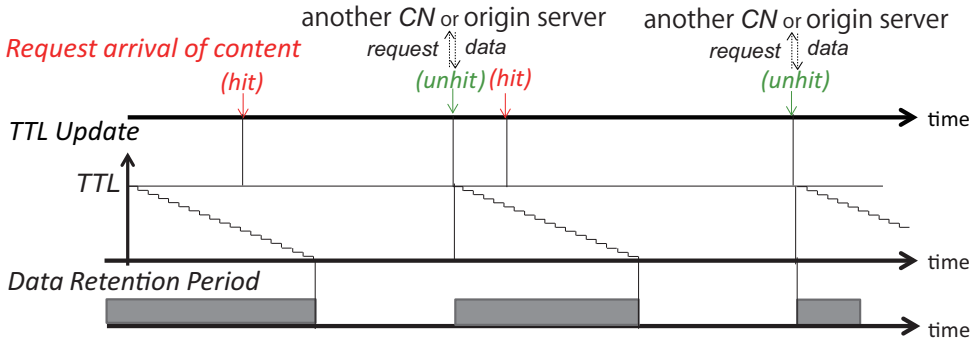
- *Policy 2:* Each *CN* resets the time counter to the TTL value for each content only when there are unsuccessful requests and decreases the counter by 1 every time unit (cf. Fig. 5.1(b)).

In both mechanisms, each *CN* judges whether to cache data by using a counter to manage TTL of content. Each *CN* discards the content with expired TTL and forwards new requests for the discarded content to another *CN* or the origin server. After receiving the delivered content, it caches the data of the content again and updates the counter based on one of the following policies in TTL-based caching [11, 34, 35].

As shown in Fig. 5.2, in general distributed TTL-based cache systems, *storage cost* increases and *bandwidth cost* decreases as the TTL value of content increases. In order to reduce the total resource cost, we should consider that there is a tradeoff for the TTL value between *storage cost*



(a) *Policy 1*



(b) *Policy 2*

Figure 5.1: TTL-based caching

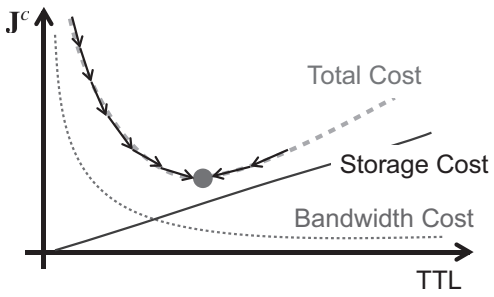


Figure 5.2: The tradeoff between *storage cost* and *bandwidth cost*

and *bandwidth cost*. Therefore, in this chapter, we propose an adaptive control mechanism to search for the TTL of each content minimizing the total resource cost in the hierarchical cache system.

While in real systems usually content objects are split into multiple chunks, our proposed

caching mechanism considers only entire content objects. However, the methodology can be extended to chunk level caching on the analogy of object level caching. Meanwhile to manage the TTL value per chunk, the system must handle much information about the TTL values. Furthermore, it is necessary to establish a new architecture for managing chunks with TTL values, which is under discussion in the research field [35, 51, 52, 49] of ICN. For instance, in the architecture for managing the TTL value per chunk, the chunks may be categorized into some classes with almost the same TTL value, or chunks of an object may have the same TTL value. The implementation of extending our mechanism to chunk level caching depends on this architecture. In this chapter, we consider the extension of the proposed mechanism to chunk level caching as out of scope and we will further study it as another work.

The remainder of this chapter is organized as follows. We next propose a system model in Section 5.2 and an adaptive TTL controller in Section 5.3. Furthermore, we present evaluation results using the proposed model in Section 5.4. Finally, we summarize this chapter in Section 5.5.

5.2 System Model

In order to model the cache characteristics in a hierarchical cache system, we first introduce the following model [11, 34, 35] of the cache probability of content c having the request rate λ^c to a *CN*.

$$\bullet \textit{ Policy 1}: \quad f(\lambda^c, TTL^c) = 1 - e^{-\lambda^c TTL^c} \tag{5.1}$$

$$\bullet \textit{ Policy 2}: \quad f(\lambda^c, TTL^c) = \frac{\lambda^c TTL^c}{1 + \lambda^c TTL^c} \tag{5.2}$$

The parameter TTL^c is the TTL value of content c. These statistical functions can provide a good approximation of the cache hit ratio of content at a *CN* under the assumption that the inter-arrival time of content requests follows an exponential distribution.

5.2.1 Request Propagation Model

In Chapter 4, we proposed an analytical model using *Policy 1* of the hierarchical cache system in which each *CN* caches content data delivered by another *CN* or an origin server when the TTL counter is above 0 and discards the content when the counter becomes 0.

The proposed model is expressed by simple matrix equations combining the statistical model of each *CN* in Eqs. (5.1) or (5.2) and can predict the cache characteristics and resource usage in hierarchical caching when the TTL value of content is given. Here, we show the evaluation model to analyze the cache characteristics of the hierarchical cache system based on *Policy 1* and *Policy 2* in TTL-based caching. The propagation of each request of content c on its caching hierarchy is expressed by the following model.

$$\mathbf{\Lambda}^c[s+1] = \mathrm{D}^c[s] \cdot \mathbf{\Lambda}^c[s] + \mathbf{R}^c, \forall c \tag{5.3}$$

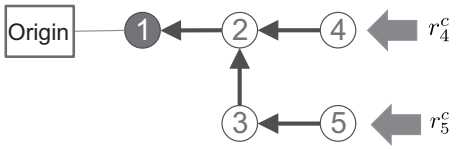
Under the condition that M, N, and s are the number of *CN*s, the number of sites having requesting users, and the number of steps that each request propagates to the next *CN*, respectively, we define $\mathbf{\Lambda}^c$ as the $M \times N$ matrix consisting of the request rates $\lambda^c_{(i,j)}$ of content c forwarded from the requesting user in site j to CN_i and $\mathbf{R^c}$ as the $M \times N$ matrix of which elements are the request rates r^c_i of content c input directly from users in site j to CN_i.

We define D^c as the $M \times M$ matrix of request propagations for content c. The (m,n) element of the matrix D^c if CN_m is a parent node of CN_n is defined as:

$$[\mathrm{D^c}]_{mn} := 1 - f(\sum_{j}^{N} \lambda^c_{(n,j)}, TTL^c) \tag{5.4}$$

otherwise, $[\mathrm{D^c}]_{mn} := 0$.

In Fig. 5.3, we show an example of these matrices for the delivery tree having 5 nodes and origin 1. With Eq. (5.3), we can predict the request propagation process for content requested from each site. Moreover, the steady state of propagated request rates per content can be derived by iteratively calculating the equation s_{max}-times, which is the maximum number of hops from each site having requesting users to its origin site.



$$\mathbf{\Lambda}^c[s] = \begin{bmatrix} \lambda^c_{(1,4)}[s] & \lambda^c_{(1,5)}[s] \\ \lambda^c_{(2,4)}[s] & \lambda^c_{(2,5)}[s] \\ \lambda^c_{(3,4)}[s] & \lambda^c_{(3,5)}[s] \\ \lambda^c_{(4,4)}[s] & \lambda^c_{(4,5)}[s] \\ \lambda^c_{(5,4)}[s] & \lambda^c_{(5,5)}[s] \end{bmatrix} \quad \mathrm{R}^c = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ r_4^c & 0 \\ 0 & r_5^c \end{bmatrix}$$

$$\mathrm{D}^c = \begin{bmatrix} 0 & 1-\boldsymbol{f}(\sum_{j=4}^5 \lambda^c_{(2,j)}, TTL^c) & 0 & 0 & 0 \\ 0 & 0 & 1-\boldsymbol{f}(\sum_{j=4}^5 \lambda^c_{(3,j)}, TTL^c) & 1-\boldsymbol{f}(\sum_{j=4}^5 \lambda^c_{(4,j)}, TTL^c) & 0 \\ 0 & 0 & 0 & 0 & 1-\boldsymbol{f}(\sum_{j=4}^5 \lambda^c_{(5,j)}, TTL^c) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 5.3: An example of matrices $\mathbf{\Lambda}^c$, $\mathbf{R}^c$, and D^c for the request propagation on the delivery tree having origin 1

5.2.2 Resource Cost Model

We next define the resource cost $\mathrm{J}^c(TTL^c, \lambda^c_{(m,n)})$ of content c in the hierarchical cache system when the TTL value is set to TTL^c as follows.

$$\begin{aligned} &\mathrm{J}^c(TTL^c, \lambda^c_{(m,n)}) \\ &= \theta_c C_s[1,\cdots,1]_{1\times M} \begin{bmatrix} f(\sum_j^N \lambda^c_{(1,j)}, TTL^c) \\ \vdots \\ f(\sum_j^N \lambda^c_{(M,j)}, TTL^c) \end{bmatrix} + \theta_c C_b[1,\cdots,1]_{1\times N} \begin{bmatrix} Tr_1^c \\ \vdots \\ Tr_N^c \end{bmatrix}. \end{aligned} \tag{5.5}$$

The first and the second terms are the total cost for storing and transmitting data of content c in the network, which use θ_c, C_s, and C_b as data size of content c [bit], the *storage cost* [cost/bit] and *bandwidth cost* [cost/bit], respectively.

The cumulative number of traffic flows Tr_j^c through each *CN* on the delivery route for content c having origin site o requested by users in site j is shown as follows.

$$\begin{aligned} \mathbf{Tr}^c &:= [Tr_1^c \cdots Tr_j^c \cdots Tr_N^c]^{\mathrm{T}} \\ &= (\mathbf{Hp} * \mathbf{\Lambda}^c)^{\mathrm{T}} \begin{bmatrix} f(\sum_j^N \lambda^c_{(1,j)}, TTL^c) \\ \vdots \\ f(\sum_j^N \lambda^c_{(M,j)}, TTL^c) \end{bmatrix} + (\mathbf{Hp}[o,] * \mathbf{\Lambda}^c[o,])^{\mathrm{T}} (1 - f(\sum_j^N \lambda^c_{(o,j)}, TTL^c)) \end{aligned} \tag{5.6}$$

Here, we define "$*$" as the element-wise product (Hadamard product) of a matrix or vector and $\mathbf{Hp} = [h_{(i,j)}]_{M\times N}$ as the matrix consisting of shortest hop length $h_{(i,j)}$ from CN_i to CN_j and the terms $\mathbf{Hp}[o,]$ and $\mathbf{\Lambda}^c[o,]$ are the o-th row vectors in matrices of $\mathbf{Hp}$ and $\mathbf{\Lambda}^c$, respectively. Moreover, the second term in Eq. (5.6) presents the amount of transmitted data which are not cached on the network.

5.3 Proposed TTL Controller

In this chapter, we propose a TTL controller which adaptively updates TTL of content to reduce the total resource cost in the hierarchical cache system.

On the assumption that the control interval for TTL is set to Δ sec (e.g. 10 sec), the proposed controller updates TTL of content c at each node by a manipulated value δt_c (e.g. 1 sec), in time unit k to reduce the total resource cost in Eq. (5.5) as Eq. (5.7).

$$TTL^c_{k+1} = \Theta(TTL^c_k + \phi(\delta t_c)) \tag{5.7}$$

The terms $\phi(\cdot)$ and $\Theta(\cdot)$ are a manipulation function and a saturation function using the maximum TTL value as follows.

$$\phi(\delta t_c) = -\mathrm{sgn}\left(\frac{d\mathrm{J}^c}{dTTL}\right)\delta t_c$$
$$\Theta(x) = \max(0, \min(\mathrm{maxTTL}, x))$$

The term $\mathrm{sgn}(\cdot)$ is the sign function. Here, the cost function of J^c is downwardly convex over the TTL value and the proposed controller searches for the TTL value of content minimizing the total cost by decreasing slope of the differential $d\mathrm{J}^c/dTTL^c$.

5.3.1 Design Algorithm of TTL Controller

In the proposed control mechanism as shown in Fig. 5.4, the TTL controller of content is designed by the following two steps.

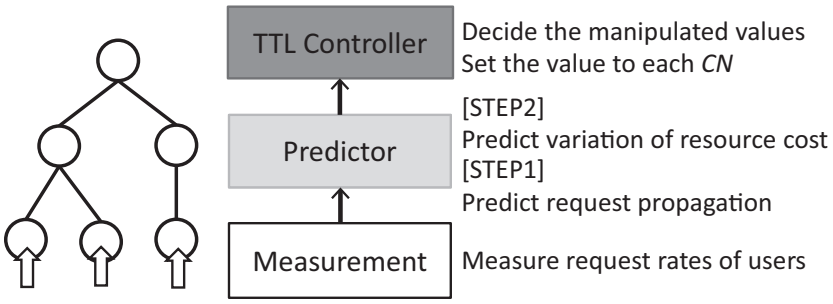


Figure 5.4: Outline of the proposed control mechanism

- **STEP1:** Predict the equilibrium of propagated request rates at each *CN* in Eq. (5.3) and the variation of the propagated requests when the TTL value of content is increased by δt_c.

- **STEP2:** Predict the variation of the resource cost when the TTL value of content is increased by δt_c, which is the differential of the cost function derived from the predictive results in **STEP1**.

All variables in the proposed model are summarized in Table 5.1. Next, we show the design process of the proposed TTL controller.

Prediction of Equilibrium and Variation of Propagated Request Rates

The equilibrium of propagated requests $\mathbf{\Lambda}^c$ of content c to each *CN* on its caching hierarchy satisfies the following matrix equation:

$$\Lambda^c = \mathrm{D}^c \mathbf{\Lambda}^c + \mathbf{R}^c, \forall c \tag{5.8}$$

where D^c is the equilibrium of the request propagation matrix for content c in TTL^c. These can be derived by iterative calculation of Eq. (5.3).

Table 5.1: Variables used in the proposed model

Variable	Type	Definition
M	given	number of *CN*s
N	given	number of sites having requesting users
θ_c	given	data size of content c [bit]
C_s	given	storage cost [cost/bit]
C_b	given	data transmission cost [cost/bit]
Hp	given	matrix consisting of hop length $h_{(i,j)}$ from CN_i to CN_j
r_i^c	measured	request rate of content c requested by users in site i
TTL^c	control	TTL of content c
δt_c	given	manipulated value of TTL control
maxTTL	given	maximum value of TTL
$\lambda_{(i,j)}^c$	predicted	propagated request rates to CN_i for content c requested by users in site j
$\delta\lambda_{(i,j)}^c$	predicted	variation of propagated request rates to CN_i for content c requested by users in site j
Tr_j^c	predicted	cumulative number of traffic flows through each *CN* on the delivery route for content c requested by users in site j
$\omega_{(i,j)}^c$	predicted	request rates for content c requested by users in site j forwarding from CN_i to its parent *CN*
$\delta\omega_{(i,j)}^c$	predicted	variation of request rates for content c requested by users in site j forwarding from CN_i to its parent *CN*

Furthermore, we predict the variation of the propagated request rates $\delta\mathbf{\Lambda}^c$ when TTL_k^c is increased by δt_c as follows.

$$\begin{aligned}&\mathbf{\Lambda}^c + \delta\mathbf{\Lambda}^c = (\mathrm{D}^c + \delta\mathrm{D}^c)(\mathbf{\Lambda}^c + \delta\mathbf{\Lambda}^c) + \mathbf{R}^c, \forall c\\ \Leftrightarrow\ &\delta\mathbf{\Lambda}^c \approx \mathrm{D}^c\delta\mathbf{\Lambda}^c + \delta\mathrm{D}^c\mathbf{\Lambda}^c, \forall c\end{aligned}$$

The (m,n) element of the matrix $\delta\mathrm{D}^c$ if CN_m is a parent node of CN_n is defined as follows.

$$[\delta\mathrm{D}^c]_{mn} := -\frac{\partial f(\sum_j^N \lambda_{(n,j)}^c, TTL^c)}{\partial TTL}\delta t - \sum_h^N \frac{\partial f(\sum_j^N \lambda_{(n,j)}^c, TTL^c)}{\partial \lambda_{(n,h)}^c}\delta\lambda_{(n,h)}^c$$

Otherwise, $[\delta \mathrm{D}^c]_{mn} := 0$. The matrix $\delta \boldsymbol{\Lambda}^c$ is defined as

$$\delta \boldsymbol{\Lambda}^c := [\delta \lambda^c_{(i,j)}]_{M \times N}.$$

As a result, we can derive the following equation with the matrices G^c and H^c.

$$\delta \boldsymbol{\Lambda}^c \approx \mathrm{G}^c \delta \boldsymbol{\Lambda}^c + \mathrm{H}^c \boldsymbol{\Lambda}^c \delta t_c. \tag{5.9}$$

For the two policies in TTL-based caching, the (m,n) elements of the matrices $[\mathrm{G}^c]_{mn}$ and $[\mathrm{H}^c]_{mn}$ if CN_m is a parent node of CN_n can be derived as follows.

- *Policy 1*:

$$[\mathrm{G}^c]_{mn} := \left(1 - TTL^c \sum_j^N \lambda^c_{(n,j)}\right) e^{-TTL^c \sum_j^N \lambda^c_{(n,j)}}$$
$$[\mathrm{H}^c]_{mn} := -\sum_j^N \lambda^c_{(n,j)} e^{-TTL^c \sum_j^N \lambda^c_{(n,j)}}$$

- *Policy 2*:

$$[\mathrm{G}^c]_{mn} := \frac{1}{1 + TTL^c \sum_j^N \lambda^c_{(n,j)}} - \frac{TTL^c \sum_j^N \lambda^c_{(n,j)}}{(1 + TTL^c \sum_j^N \lambda^c_{(n,j)})^2}$$
$$[\mathrm{H}^c]_{mn} := -\frac{\sum_j^N \lambda^c_{(n,j)}}{(1 + TTL^c \sum_j^N \lambda^c_{(n,j)})^2}$$

In both cases, $[\mathrm{G}^c]_{mn} = 0$ and $[\mathrm{H}^c]_{mn} = 0$ if CN_m is not a parent node of CN_n.

Because the square matrix $(I - \mathrm{G}^c)$ always has full rank, we can derive the following equation.

$$\delta \boldsymbol{\Lambda}^c = (I - \mathrm{G}^c)^{\mathrm{T}} \mathrm{H}^c \boldsymbol{\Lambda}^c \delta t_c \tag{5.10}$$

Prediction of Variation of Resource Cost

For the total resource cost J^c in Eq. (5.5), we can differentiate the cost function $\delta\mathrm{J}^c(TTL^c, \delta t_c, \mathbf{\Lambda}, \delta\mathbf{\Lambda})$, which is the variation of the total resource cost J^c when TTL^c is increased by δt_c, as follows.

$$\delta\mathrm{J}^c = \frac{\partial \mathrm{J}^c}{\partial TTL}\delta t_c + \sum_{(m,n)} \frac{\partial \mathrm{J}^c}{\partial \lambda^c_{(m,n)}}\delta\lambda^c_{(m,n)} \tag{5.11}$$

$$= \theta_c C_s[1,..,1]_{1\times M}\left(\begin{bmatrix} \frac{\partial f(\sum_j^N \lambda^c_{(1,j)}, TTL^c)}{\partial TTL} \\ \vdots \\ \frac{\partial f(\sum_j^N \lambda^c_{(M,j)}, TTL^c)}{\partial TTL} \end{bmatrix}\delta t_c + \left(\begin{bmatrix} \frac{\partial f(\sum_j^N \lambda^c_{(1,j)}, TTL^c)}{\partial \lambda^c_{(1,1)}} .. \frac{\partial f(\sum_j^N \lambda^c_{(1,j)}, TTL^c)}{\partial \lambda^c_{(1,N)}} \\ \vdots \\ \frac{\partial f(\sum_j^N \lambda^c_{(M,j)}, TTL^c)}{\partial \lambda^c_{(M,1)}} .. \frac{\partial f(\sum_j^N \lambda^c_{(M,j)}, TTL^c)}{\partial \lambda^c_{(M,N)}} \end{bmatrix} * \delta\mathbf{\Lambda}^c\right)\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}_{N\times 1}\right)$$

$$+\theta_c C_b[1,..,1]_{1\times N}(\mathbf{Hp} * \delta\mathbf{\Lambda}^c)^{\mathrm{T}}\begin{bmatrix} f(\sum_j^N \lambda^c_{(1,j)}, TTL^c) \\ \vdots \\ f(\sum_j^N \lambda^c_{(M,j)}, TTL^c) \end{bmatrix}$$

$$+\theta_c C_b[1,..,1]_{1\times N}(\mathbf{Hp} * \mathbf{\Lambda}^c)^{\mathrm{T}}\left(\begin{bmatrix} \frac{\partial f(\sum_j^N \lambda^c_{(1,j)}, TTL^c)}{\partial TTL} \\ \vdots \\ \frac{\partial f(\sum_j^N \lambda^c_{(M,j)}, TTL^c)}{\partial TTL} \end{bmatrix}\delta t_c + \left(\begin{bmatrix} \frac{\partial f(\sum_j^N \lambda^c_{(1,j)}, TTL^c)}{\partial \lambda^c_{(1,1)}} .. \frac{\partial f(\sum_j^N \lambda^c_{(1,j)}, TTL^c)}{\partial \lambda^c_{(1,N)}} \\ \vdots \\ \frac{\partial f(\sum_j^N \lambda^c_{(M,j)}, TTL^c)}{\partial \lambda^c_{(M,1)}} .. \frac{\partial f(\sum_j^N \lambda^c_{(M,j)}, TTL^c)}{\partial \lambda^c_{(M,N)}} \end{bmatrix} * \delta\mathbf{\Lambda}^c\right)\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}_{N\times 1}\right)$$

$$+\theta_c C_b[1,..,1]_{1\times N}\left((\mathbf{Hp}[o,] * \delta\mathbf{\Lambda}^c[o,])^{\mathrm{T}}(1 - f(\sum_j^N \lambda^c_{(o,j)}, TTL^c))\right.$$

$$\left. -(\mathbf{Hp}[o,] * \mathbf{\Lambda}^c[o,])^{\mathrm{T}}\left(\frac{\partial f(\sum_j^N \lambda^c_{(o,j)}, TTL^c)}{\partial TTL}\delta t_c + \left(\begin{bmatrix} \frac{\partial f(\sum_j^N \lambda^c_{(o,j)}, TTL^c)}{\partial \lambda^c_{(1,1)}} .. \frac{\partial f(\sum_j^N \lambda^c_{(o,j)}, TTL^c)}{\partial \lambda^c_{(1,N)}} \end{bmatrix} * \delta\mathbf{\Lambda}^c[o,]\right)\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}_{N\times 1}\right)\right)$$

By substituting TTL^c_k and $\mathbf{\Lambda}^c$ in Eq. (5.8), and $\delta\mathbf{\Lambda}^c$ in Eq. (5.10) for the differential function $\delta\mathrm{J}^c$, we can derive the following linear equation of δt_c where Q^c is a scalar.

$$\delta\mathrm{J}^c = Q^c\delta t_c$$

Therefore, the TTL controller in Eq. (5.7) can be derived by

$$TTL^c_{k+1} = \Theta(TTL^c_k + \phi(\delta t_c)), \tag{5.12}$$
$$\phi(\delta t_c) = -\mathrm{sgn}(Q^c)\delta t_c,$$

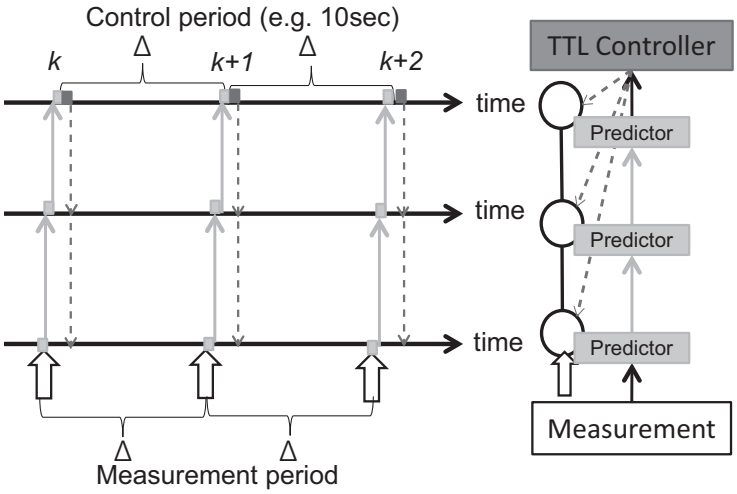


Figure 5.5: Distributed prediction-based control mechanism

which means that the TTL value of content is updated at each *CN* by the following rule. When Q^c is negative/positive, TTL^c is increased/decreased by δt_c and when Q^c is zero, TTL^c is not updated.

The calculation order per content in the proposed prediction process is $O(M^2N)$, but we can divide the predictive process into local prediction processes of each *CN* with calculation order of $O(N)$.

5.3.2 Distributed Prediction

The proposed algorithm of the TTL controller can be divided into recursive processes at each *CN* which coordinates *CN*s on the delivery tree every time unit Δ (cf., Fig. 5.5).

In the proposed mechanism, the tracking performance to change the request rates becomes better as the control interval Δ is smaller. However, the control interval should be set as a long period to measure the statistics of request rates. Meanwhile, Santos *et al.* [57] proposed a statistics table for CCN, which can online estimate request rates by using the exponentially weighted moving average (EWMA), to implement their proposed Content-Centric Dissemination Algorithm (CEDO) for maximizing the total delivery throughput. Therefore, we can also use EWMA to online measure the request rates. In this chapter, we assume that the exchange of information between *CN*s and TTL controllers, including measured request rates and manipulated TTL values for each content, is

bundled in a single measurement/control message.

In the distributed mechanism as shown in Fig. 5.6, CN_m predicts the propagated request rate $\mathbf{\Lambda}_m^c = \lambda_{(m,j)}^c, \forall j,c$ in Eq. (5.8), the variation of request rate $\delta\mathbf{\Lambda}_m^c = \delta\lambda_{(m,j)}^c, \forall j,c$ in Eq. (5.10), and the variation of the resource cost $\delta \mathrm{J}_m^c$ in Eq. (5.13) in the order from *CN*s (leaves) on the bottom layer to a *CN* (root) on the upper layer in the delivery tree. The variation of the resource cost $\delta \mathrm{J}_m^c$ of CN_m in Eq. (5.13) is derived by dividing the differential cost $\delta \mathrm{J}^c$ in Eq. (5.11).

$$
\begin{aligned}
\delta \mathrm{J}_m^c &= \theta_c C_s \left(\frac{\partial f(\sum_j^N \lambda_{(m,j)}^c, TTL^c)}{\partial TTL} \delta t_c + \sum_h^N \frac{\partial f(\sum_j^N \lambda_{(m,j)}^c, TTL^c)}{\partial \lambda_{(m,h)}^c} \delta \lambda_{(m,h)}^c \right) \\
&+ \theta_c C_b \left(\sum_j^N h_{(m,j)} \delta \lambda_{(m,j)}^c f(\sum_j^N \lambda_{(m,j)}^c, TTL^c) \right. \\
&\left. + \sum_j^N h_{(m,j)} \lambda_{(m,j)}^c \left(\frac{\partial f(\sum_j^N \lambda_{(m,j)}^c, TTL^c)}{\partial TTL} \delta t_c + \sum_h^N \frac{\partial f(\sum_j^N \lambda_{(m,j)}^c, TTL^c)}{\partial \lambda_{(m,h)}^c} \delta \lambda_{(m,h)}^c \right) \right) \qquad (5.13) \\
&+ \begin{cases} 0 & \text{if } m \neq o \\ \theta_c C_b \left(\sum_j^N h_{m,j} \delta \lambda_{m,j}^c \left(1 - f(\sum_j^N \lambda_{(m,j)}^c, TTL^c) \right) \right. & \\ \left. - \sum_j^N h_{m,j} \lambda_{m,j}^c \left(\frac{\partial f(\sum_j^N \lambda_{(m,j)}^c, TTL^c)}{\partial TTL} \delta t_c + \sum_h^N \frac{\partial f(\sum_j^N \lambda_{(m,j)}^c, TTL^c)}{\partial \lambda_{(m,h)}^c} \delta \lambda_{(m,h)}^c \right) \right) & \text{if } m = o \end{cases}
\end{aligned}
$$

Prediction of Equilibrium and Variation of Propagated Request Rates

CN_m first calculates the information Ω_m^c, which is the predictive amount of propagated request rates from CN_m to its parent CN_p, by the product of the matrix element $\mathrm{D}^c[p,m]$ and the m-th row vector Λ_m^c in the right side of Eq. (5.8).

$$
\Omega_m^c = \left(1 - f(\sum_j^N \lambda_{(m,j)}^c, TTL^c) \right) \Lambda_m^c
$$

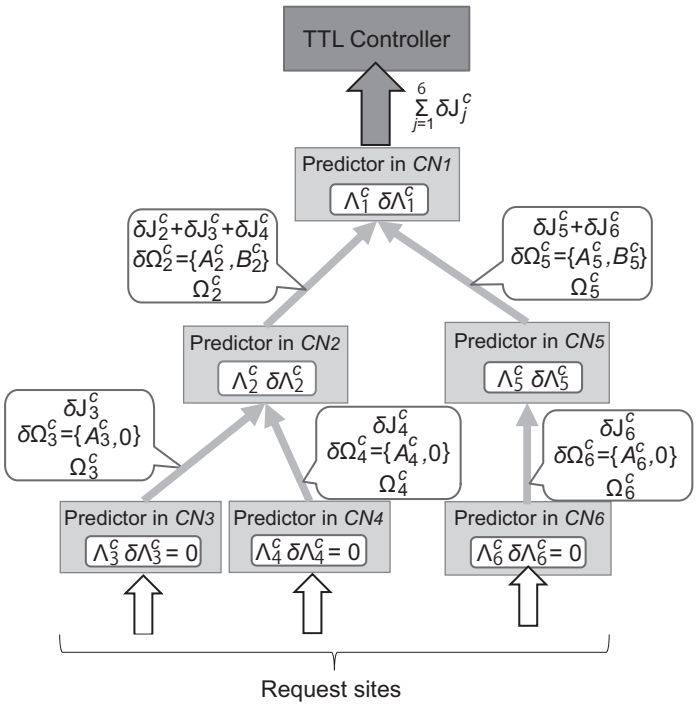


Figure 5.6: An example of the distributed prediction

Furthermore, CN_m can predict the equilibrium of propagated request rates Λ_m^c by using its children information on the delivery tree,

$$\Lambda_m^c = \sum_{h \in \mathrm{Child}_m} \Omega_h^c + \mathrm{R}_m^c \tag{5.14}$$

where R_m^c is the request rate input directly to CN_m by users and corresponds to the m-th row vector of matrix $\mathbf{R}^c$ in Eq. (5.8). The terms Ω_m^c and Λ_m^c are both $1 \times N$ vectors and Child_m is a set of CNs directly under CN_m.

Furthermore, CN_m calculates $\delta\Omega_m^c$, which is the variation of the request rate from CN_m to its parent CN_p when increasing TTL by δt_c. It is calculated by the sum of the product of the matrix element $\mathrm{G}^c[p,m]$ and vector $\delta\Lambda_m^c$ and the product of the matrix element $\mathrm{H}^c[p,m]$ and vector Λ_m^c in the right side of Eq. (5.9). Therefore, the information becomes a linear function of δt_c with A_m^c and

B_m^c having Λ_m^c in Eq. (5.14) and $\delta\Lambda_m^c$ derived by the children information $\delta\Omega_h^c$ of CN_m

$$\delta\Omega_m^c = A_m^c \delta t_c + B_m^c$$

where A_m^c and B_m^c are $1 \times N$ vectors which are defined for both policies as follows.

- *Policy 1*:

$$A_m^c = -\left(\sum_j^N \lambda_{(m,j)}^c e^{-TTL^c \sum_j^N \lambda_{(m,j)}^c}\right) \Lambda_m^c$$
$$B_m^c = \left(\left(1 - TTL^c \sum_j^N \lambda_{(m,j)}^c\right) e^{-TTL^c \sum_j^N \lambda_{(m,j)}^c}\right) \delta\Lambda_m^c$$

- *Policy 2*:

$$A_m^c = -\left(\frac{\sum_j^N \lambda_{(m,j)}^c}{(1 + TTL^c \sum_j^N \lambda_{(m,j)}^c)^2}\right) \Lambda_m^c$$
$$B_m^c = \left(\frac{1}{(1 + TTL^c \sum_j^N \lambda_{(m,j)}^c)^2}\right) \delta\Lambda_m^c$$

Moreover, CN_m predicts the variation of propagated request rates Λ_m^c by using its children information $\delta\Omega_h^c$,

$$\delta\Lambda_m^c = \sum_{h \in \text{Child}_m} \delta\Omega_h^c \tag{5.15}$$

where $\delta\Lambda_m^c$ and $\delta\Omega_m^c$ are both $1 \times N$ vectors.

Prediction of Variation of Resource Cost

The differential costs in Eq. (5.13) for the two policies are shown in Eqs. (5.16) and (5.17).

- *Policy 1*:

$$
\begin{aligned}
&\delta \mathrm{J}^c_m(TTL^c, \delta t_c, \mathbf{\Lambda}^c_m, \delta \mathbf{\Lambda}^c_m) \\
&= \theta_c C_s \left(\sum_j^N \lambda^c_{(m,j)} e^{-TTL^c \sum_j^N \lambda^c_{(m,j)}} \delta t_c + TTL^c e^{-TTL^c \sum_j^N \lambda^c_{(m,j)}} \sum_h^N \delta \lambda^c_{(m,h)} \right) \\
&+ \theta_c C_b \left(\sum_j^N h_{(m,j)} \delta \lambda^c_{(m,j)} (1 - e^{-TTL^c \sum_j^N \lambda^c_{(m,j)}}) \right. \\
&\left. + \sum_j^N h_{(m,j)} \lambda^c_{(m,j)} \left(\sum_j^N \lambda^c_{(m,j)} e^{-TTL^c \sum_j^N \lambda^c_{(m,j)}} \delta t_c + TTL^c e^{-TTL^c \sum_j^N \lambda^c_{(m,j)}} \sum_h^N \delta \lambda^c_{(m,h)} \right) \right) \\
&+ \begin{cases} 0 & \text{if } m \neq o \\ \theta_c C_b \left(\sum_j h_{m,j} \delta \lambda^c_{m,j} e^{-TTL^c \sum_j^N \lambda^c_{(m,j)}} - \sum_j^N h_{m,j} \lambda^c_{m,j} \left(\sum_j^N \lambda^c_{(m,j)} e^{-TTL^c \sum_j^N \lambda^c_{(m,j)}} \delta t_c \right. \right. & \\ \left. \left. + TTL^c e^{-TTL^c \sum_j^N \lambda^c_{(m,j)}} \sum_h^N \delta \lambda^c_{(m,h)} \right) \right) & \text{if } m = o \end{cases}
\end{aligned}
\tag{5.16}
$$

- *Policy 2*:

$$
\begin{aligned}
&\delta \mathrm{J}^c_m(TTL^c, \delta t_c, \mathbf{\Lambda}^c_m, \delta \mathbf{\Lambda}^c_m) \\
&= \theta_c C_s \left(\frac{\sum_j^N \lambda^c_{(m,j)}}{(1 + TTL^c \sum_j^N \lambda^c_{(m,j)})^2} \delta t_c + \frac{TTL^c}{(1 + TTL^c \sum_j^N \lambda^c_{(m,j)})^2} \sum_h^N \delta \lambda^c_{(m,h)} \right) \\
&+ \theta_c C_b \left(\sum_j^N h_{(m,j)} \delta \lambda^c_{(m,j)} \frac{TTL^c \sum_j^N \lambda^c_{(m,j)}}{1 + TTL^c \sum_j^N \lambda^c_{(m,j)}} \right. \\
&\left. + \sum_j^N h_{(m,j)} \lambda^c_{(m,j)} \left(\frac{\sum_j^N \lambda^c_{(m,j)}}{(1 + TTL^c \sum_j^N \lambda^c_{(m,j)})^2} \delta t_c + \frac{TTL^c}{(1 + TTL^c \sum_j^N \lambda^c_{(m,j)})^2} \sum_h^N \delta \lambda^c_{(m,h)} \right) \right) \\
&+ \begin{cases} 0 & \text{if } m \neq o \\ \theta_c C_b \left(\sum_j^N h_{m,j} \delta \lambda^c_{m,j} \frac{1}{1 + TTL^c \sum_j^N \lambda^c_{(m,j)}} - \sum_j^N h_{m,j} \lambda^c_{m,j} \left(\frac{\sum_j^N \lambda^c_{(m,j)}}{(1 + TTL^c \sum_j^N \lambda^c_{(m,j)})^2} \delta t_c \right. \right. & \\ \left. \left. + \frac{TTL^c}{(1 + TTL^c \sum_j^N \lambda^c_{(m,j)})^2} \sum_h^N \delta \lambda^c_{(m,h)} \right) \right) & \text{if } m = o \end{cases}
\end{aligned}
\tag{5.17}
$$

Table 5.2: Power cost parameters

Device (Product)	Power / Spec	Power Cost per 1 sec
Memory (DRAM)	10 W / 4 GB	$C_s = 3.125 \times 10^{-10}$ J/bit
Node (CRS-1)	4185 W / 320 Gbps	$C_b = 1.3 \times 10^{-8}$ J/bit

By substituting TTL_k^c, Λ_m^c in Eq. (5.14), and $\delta\Lambda_m^c$ in Eq. (5.15) for these models, $\delta \mathrm{J}_m^c$ can be expressed as a linear function of δt_c. Furthermore, CN_m informs its parent *CN* of the cumulative differential cost as follows.

$$\delta \mathrm{J}_m^c + \sum_{h \in \mathrm{L}_m} \delta \mathrm{J}_h^c$$

where L_m is a set of *CN*s below CN_m on the delivery tree.

Finally, the TTL controller on the root *CN* can calculate the sum of differential costs $\sum_{m=1}^{M} \delta \mathrm{J}_m^c$ as $Q_c \delta t_c$ in Eq. (5.12). As a result, the proposed TTL controller can decide the manipulated value δt_c of TTL.

5.4 Evaluation

We evaluate the effectiveness of the proposed mechanism when controlling the TTL value of content based on the predictive models. The evaluation conditions are set to the following.

- **Test network:** 4-layered tree topology with in total 85 nodes of which 64 nodes are edge nodes having requesting users, cf. Fig. 5.7. The root node has an origin server. The memory size of each *CN* is set to sufficient size to cache 10^3 objects.

- **Content information:** Zipf-distributed requests from each site i for $K = 10^3$ objects are defined as $r_i^c = \gamma k^{-\alpha}/c$, $c = \sum_{k=1}^{K} k^{-\alpha}$, cf. Fig. 5.8. We set α to 0.8 for User Generated Content (UGC) and 1.2 for VoD [43, 42, 26] and γ to 100 [requests/sec]. For analytical simplicity, we assume that each content object has the same size of θ_c which is set to 1.

- **Cost parameters:** The cost ratios between *storage cost* C_s and *bandwidth cost* C_b are set

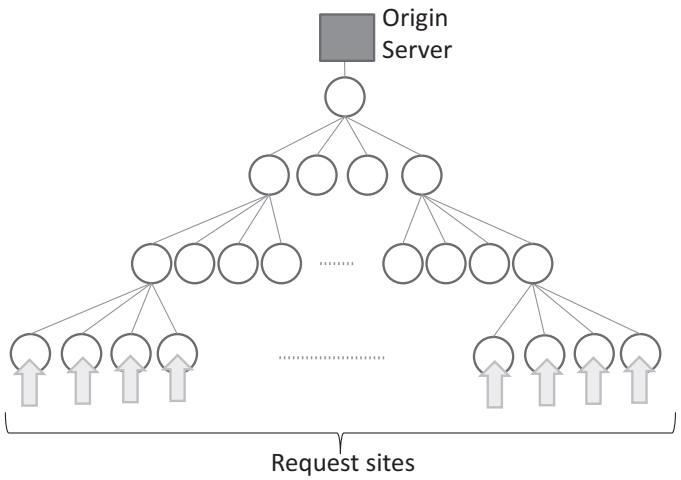


Figure 5.7: Tree topology for evaluation

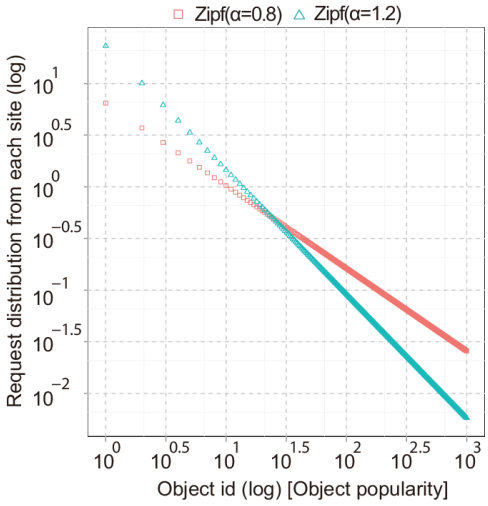


Figure 5.8: Request distribution from each site

to various values as 0.03 : 0.97, 0.1 : 0.9, 0.5 : 0.5, and 0.9 : 0.1. Here, the cost ratio of 0.03 : 0.97 corresponds to the energy cost as shown in Table.5.2

- **Control parameters:** We assume the manipulated value δt_c as 1 sec. Moreover, initial TTLs of content and the maximum TTL are set to 1 sec and 300 sec, respectively.

Table 5.3: Comparison between numerical evaluation and simulation

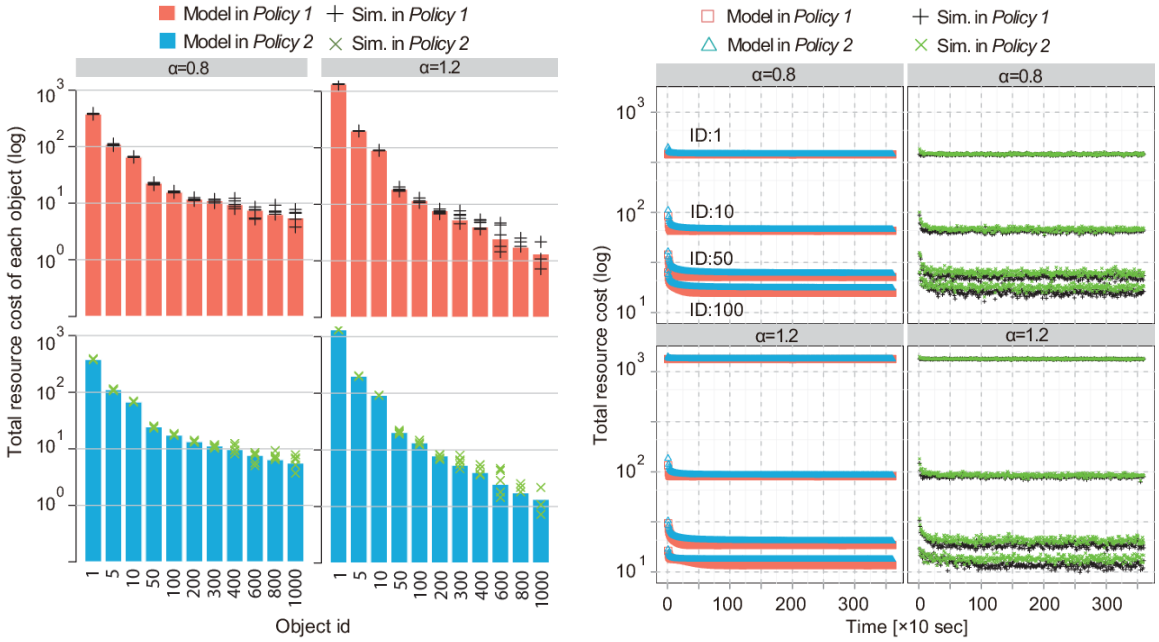
Process		Numerical Calculation	Simulation
input	request rate r_i^c	given by a static value	measured by each edge *CN*
cache system	cache probability	calculated by Eqs. (5.1), (5.2) and controlled TTLs	measured by each *CN*
	request propagation and data delivery	calculated by Eqs. (5.3) and (5.6)	simulated by packet process
output	total cost J^c	calculated by Eq. (5.5) and controlled TTLs	measured by each *CN*

5.4.1 Comparison of the Proposed Model and Simulation

To verify the validity of the model proposed in Sect. 5.2, we first compare control results calculated by the proposed model with that measured by simulations for 11 representative objects with $\alpha = 0.8$ and $\alpha = 1.2$ in $C_s : C_b = 0.1 : 0.9$.

Here, we show the difference between the numerical calculation based on the proposed model and the packet simulation in Table 5.3. Both evaluations are executed by the distributed mechanism in Subsect. 5.3.2 and TTL control in Eq. (5.12). Meanwhile in the numerical evaluation, the request rate r_i^c as input is set to a static value and the total cost J^c as output is calculated by Eq. (5.5) and the controlled TTL value. In the simulation, the request rate r_i^c is measured as the number of requests by each edge *CN* and the total cost J^c is calculated as the sum of *storage cost* and *bandwidth cost* measured by each *CN*. Furthermore, the request propagation and data delivery are simulated as packet processes and we set the control interval Δ and simulation time as 10 sec and 3600 sec, respectively. Moreover, to measure the average rate of user requests r_i^c within 10 sec in the same way as [57], we use EWMA in consideration of historical data in the past 600 sec.

In Fig. 5.9(a), we show the total resource cost estimated by the model and the average cost within 10 sec measured from 3560 sec to 3600 sec in the simulation. Additionally, Fig.5.9(b) shows the time change of total resource cost estimated by the proposed model and measured by simulation for 4 objects. The average cost measured by simulation varies more widely for content objects having lower request rates but the results estimated by the proposed model provide suitable



(a) Comparison of controlled cost

(b) Time change of cost for object ids: 1, 10, 50, and 100

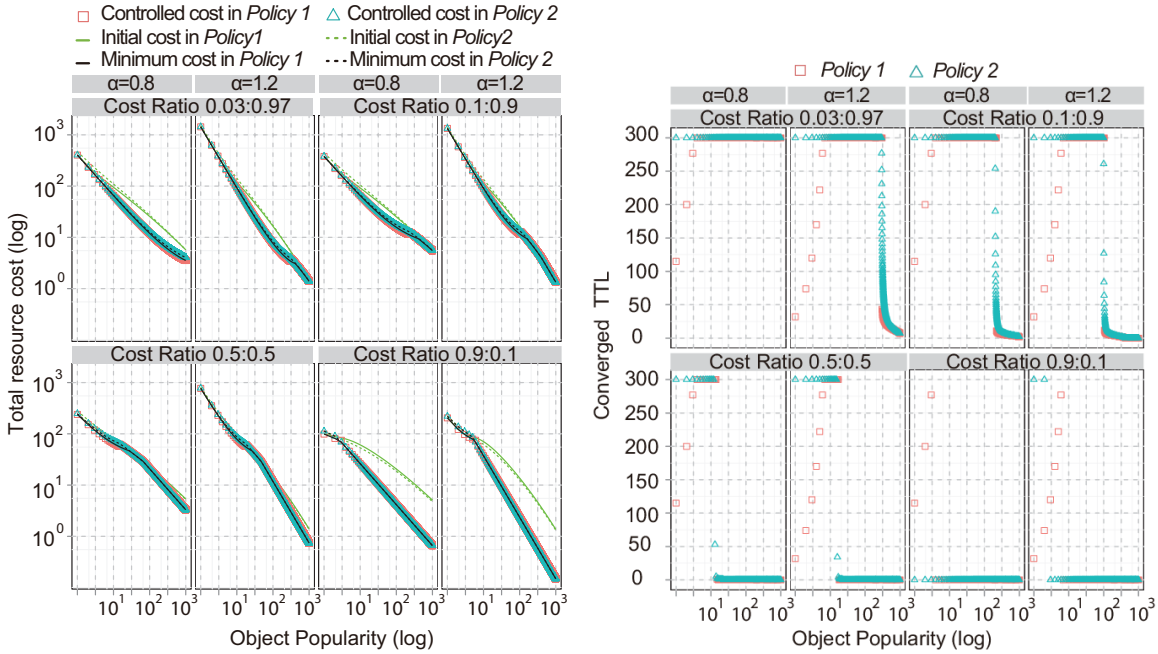
Figure 5.9: Total resource cost estimated by the proposed model and measured by simulations for different object ids in $C_s : C_b = 0.1 : 0.9$

approximations of the simulation results. Therefore, we see that the proposed model can express the statistical characteristics for the proposed control mechanism.

5.4.2 Effectiveness of the Proposed Mechanism

Next, we demonstrate the effectiveness of the proposed mechanism by using the model-based analysis. Figure 5.10(a) shows the total resource cost for each object when setting the initial TTLs to 1 sec, the controlled cost by the proposed mechanism, and the minimum cost in the given range of TTLs from 1 sec to 300 sec which are calculated offline by the request propagation model in Eq. (5.3), respectively. These results demonstrate that the proposed mechanism is able to find TTL values that reduce the resource cost to near the minimum cost under the given conditions, i.e., the control range of TTL values, initial TTL values, the policy of TTL-based caching, and cost parameters.

Figure 5.10(b) shows the converged TTLs controlled by the proposed mechanism. As a result,



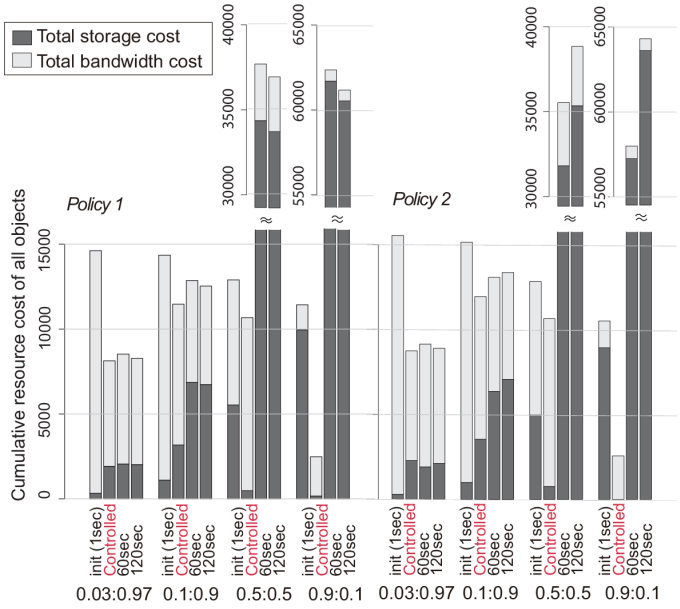
(a) Resource cost

(b) TTLs

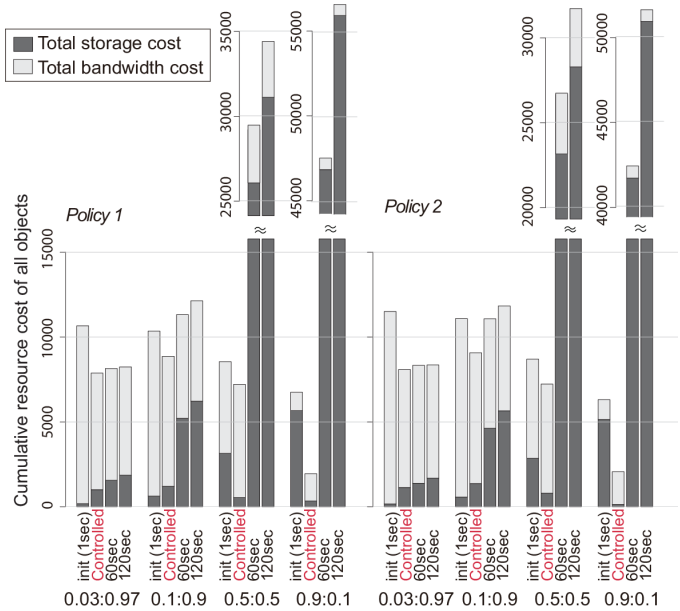
Figure 5.10: Convergence values of resource cost and TTLs for content objects

the controlled TTLs become smaller as *storage cost* becomes larger. Moreover, the TTL values of content having large request rates converge at large values which are saturated by the maximum TTL of 300 sec. Here, the TTL values in *Policy 1* from id 1 to id 10 are not saturated at 300 sec in spite of high popularity objects. This why the cache probability of Eq. (5.1) for these objects in *Policy 1* becomes 1 even when the TTL values are small.

Additionally, Fig. 5.11 presents the cumulative resource cost of all objects. To compare the effectiveness of the proposed method with conventional methods using static TTLs, these figures additionally show the results when setting static TTLs of all objects to 1 sec, 60 sec, and 120 sec. In these results, we see that the proposed mechanism can reduce the total resource cost compared to the cases when setting the TTL values of all objects to 1 sec as the initial value. Furthermore, this mechanism can locally search for TTL values which can realize lower cost than that of when the TTL values are set to static values of 60 sec and 120 sec.



(a) Objects with $\alpha = 0.8$



(b) Objects with $\alpha = 1.2$

Figure 5.11: Cumulative resource cost of all objects and the effectiveness of reducing the total resource cost (initial cost, controlled cost, cost when setting TTLs to 60 and 120 sec)

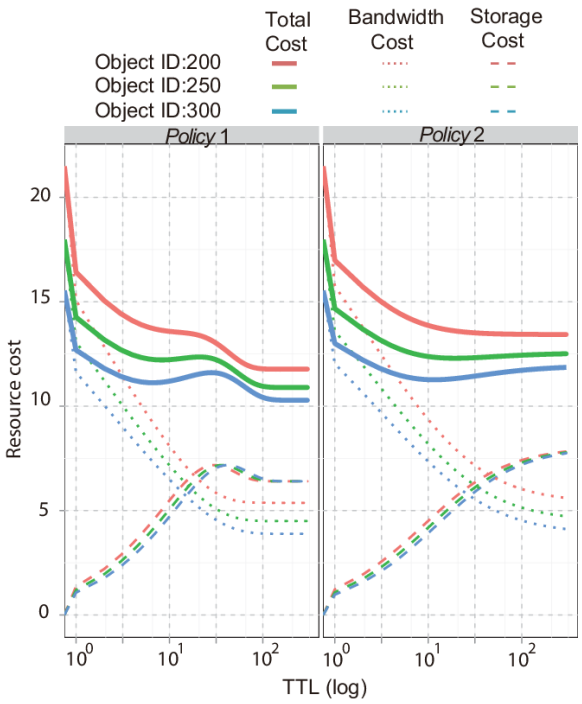
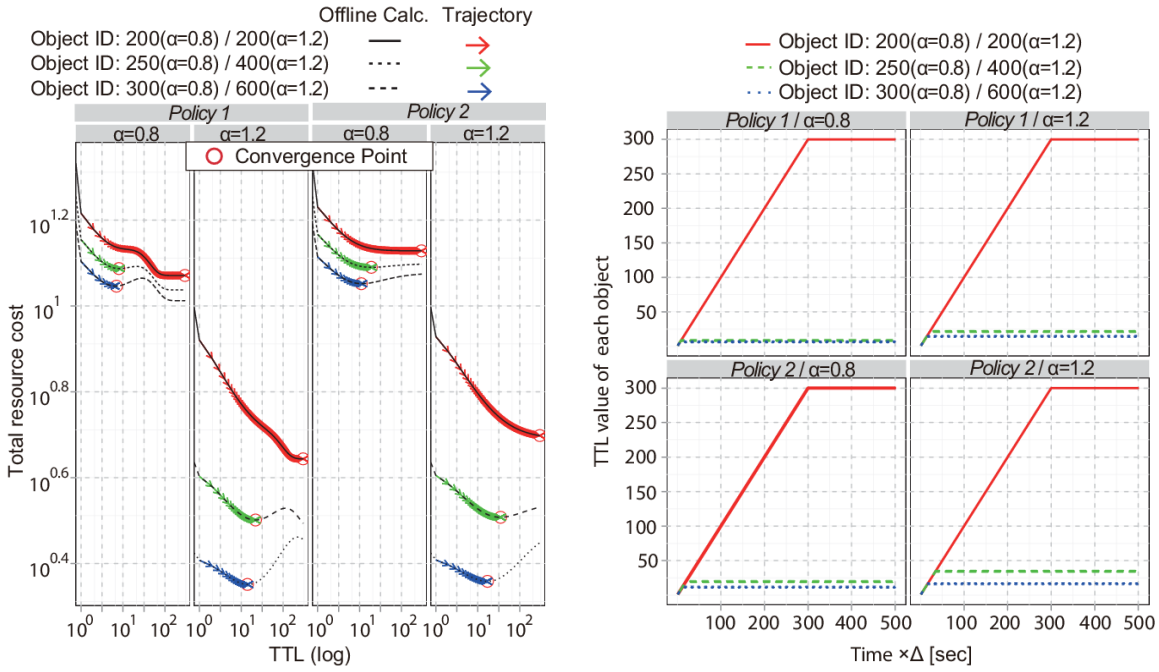


Figure 5.12: Tradeoff between storage cost and bandwidth cost for object ids: {200, 250, 300} with $\alpha = 0.8$ in $C_s : C_b = 0.1 : 0.9$

5.4.3 Verification of Optimality

Figure 5.12 presents the tradeoffs between storage cost and bandwidth cost for object ids: {200, 250, 300} with $\alpha = 0.8$ in $C_s : C_b = 0.1 : 0.9$, which are derived by the offline calculation using the request propagation model in Eq. (5.3). These results show that the total storage cost of all *CN*s in *Policy 1* is not an increasing monotonic function. This is why there is a tradeoff between the total storage cost and TTL in *Policy 1*, which means that the cache probability of the other nodes except for edge nodes approaches 0 as the cache probability of edge nodes is close to 1. As a result, it is conceivable that the curve of total resource cost depends on the request distribution and target topology.

Figure 5.13(a) presents the control trajectories of the proposed mechanism and the minimum cost derived by offline calculation for object ids: {200, 250, 300} with $\alpha = 0.8$ in $C_s : C_b = 0.1 : 0.9$ and object ids: {200, 400, 600} with $\alpha = 1.2$ in $C_s : C_b = 0.03 : 0.97$. Additionally in Fig. 5.13(b),



(a) Control trajectory

(b) Time change of TTLs

Figure 5.13: Control trajectory and time change of TTLs for objects with $\alpha = 0.8$ in $C_s : C_b = 0.1 : 0.9$ and with $\alpha = 1.2$ in $C_s : C_b = 0.03 : 0.97$ when the initial TTL is set to 1 sec

we present the change of TTL values of each content. In these results, the proposed controllers in *Policy 1* converge at the TTL value and drop to the local minimum cost. Meanwhile, the TTL controllers in *Policy 2* can search for the minimum cost in the control range of TTLs by descending the slope of the cost function.

In Fig. 5.14, we change the initial TTLs from 1 sec to 150 sec. In the results for object ids: $\{250, 300\}$ with $\alpha = 0.8$ and object id: $\{400\}$ with $\alpha = 1.2$ in *Policy 1*, the TTL values converge at different points from those when the initial TTLs are 1. However, because the request rates from users vary momentarily, we can guess that there is little probability of staying on the local minimum cost. Furthermore, in order to avoid converging to local solutions, it is effective to periodically reset the TTL values randomly.

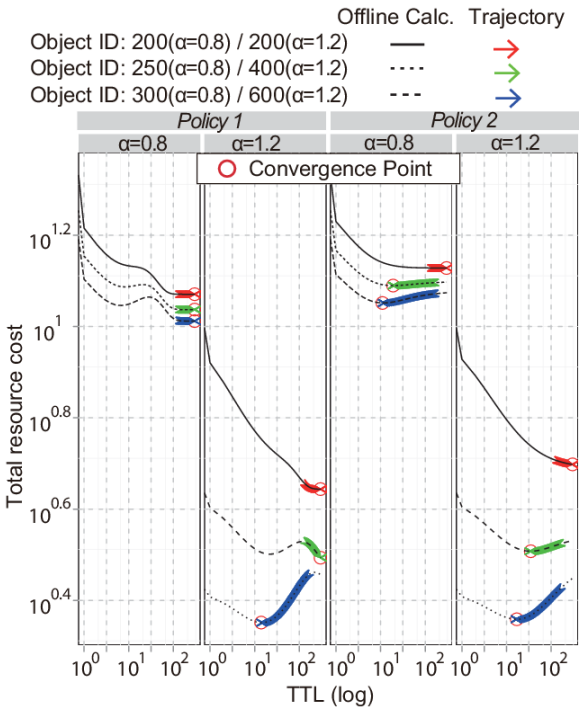


Figure 5.14: Control trajectory for object with $\alpha = 0.8$ in $C_s : C_b = 0.1 : 0.9$ and with $\alpha = 1.2$ in $C_s : C_b = 0.03 : 0.97$ when the initial TTL is set to 150 sec

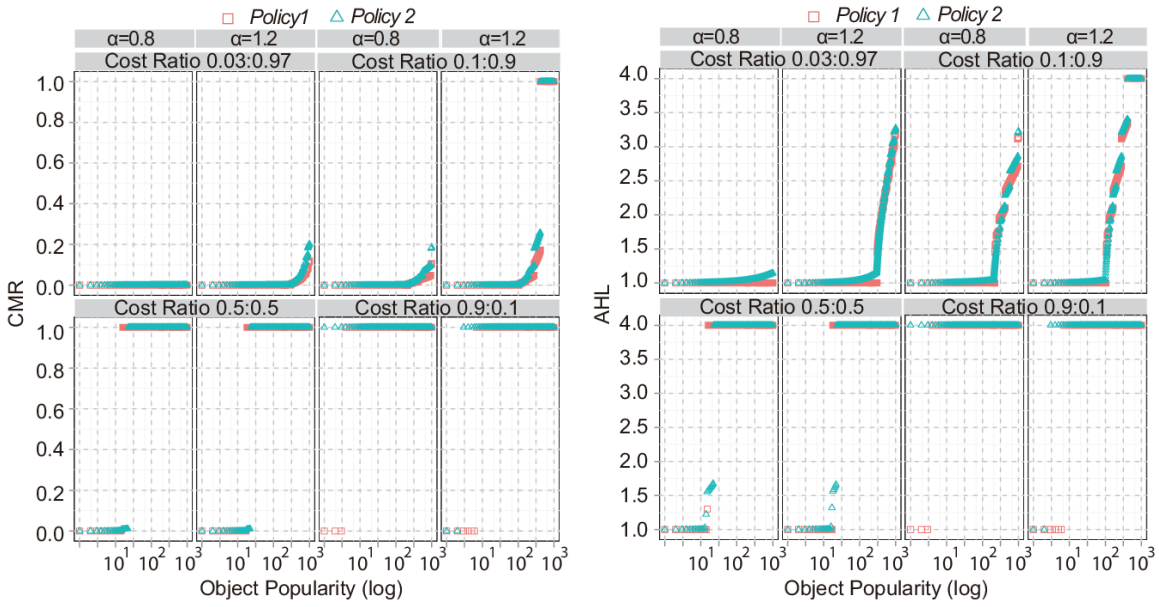
5.4.4 Impact of the TTL Control on Cache Performance

We show the cache miss ratio and average hop length in the control results derived by Fig. 5.10(a). The cache miss ratio CMR^c and the average hop length AHL^c of object c in the network can be defined as the following metrics.

$$\mathrm{CMR}^c := \frac{\sum_j^N \lambda_{(o,j)}^c \left(1 - f(\sum_k^N \lambda_{(o,k)}^c, TTL^c)\right)}{\sum_j^N r_j^c} \tag{5.18}$$

$$\mathrm{AHL}^c := \frac{\sum_j^N Tr_j^c}{\sum_j^N r_j^c} \tag{5.19}$$

Figure 5.15 shows the cache miss ratio and the average hop length for all contents when the proposed controller converges at each equilibrium. In the results, as content has lower request rates and *storage cost* becomes larger, the cache miss ratio and average hop length of content become higher and longer, respectively.



(a) Cache miss ratio

(b) Average hop length

Figure 5.15: Cache miss ratio and average hop length of all objects when converging at the minimum cost

5.4.5 Adaptability of TTL Control

We finally demonstrate the adaptability of the proposed mechanism. Figure 5.16 presents the control trajectory for object ids: $\{200, 250, 300\}$ in $C_s : C_b = 0.1 : 0.9$ when the request rate is changed from $\alpha = 0.8$ to $\alpha = 1.2$ in the middle of the cache operation. In these results, the proposed controller tracks the curve of the cost function caused by the change of the request rate of content. Meanwhile for object ids: $\{250, 300\}$, the controllers in *Policy 1* first converge at the local minimum cost for the cost curve with $\alpha = 0.8$ and can search for the minimum cost in the control range of TTLs for the cost curve with $\alpha = 1.2$ after changing the request rate of content. Therefore, we see that our proposed mechanism can adaptively search for a TTL value to reduce the total resource cost in the distributed cache system according to the change of the request distribution.

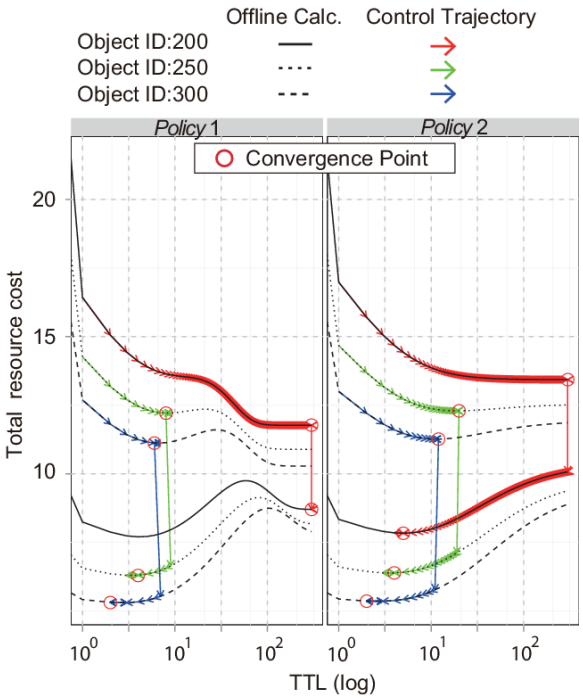


Figure 5.16: Control trajectory for object ids: $\{200, 250, 300\}$ in $C_s : C_b = 0.1 : 0.9$

5.5 Summary

In this chapter, we proposed an adaptive control mechanism which auto-tunes the TTL value of content based on predictive models to reduce the total resource cost in distributed cache systems and introduced a distributed solution. The proposed mechanism periodically decides the manipulated values of TTL by predicting the impact of the TTL values on the total resource cost in the hierarchical cache system. In the evaluations, we compared the control results estimated by the proposed model with those obtained by simulations. Furthermore, we analyzed the effectiveness of the proposed mechanism and showed that our proposed mechanism can search for a TTL value of content to reduce the total resource cost by descending the slope of the cost function. Additionally, we analyzed the cache performance in the proposed control mechanism by using performance metrics such as cache miss ratio and average hop length and finally demonstrated the adaptability of the proposed mechanism.

Chapter 6

Conclusion

In this thesis, we investigated theoretic models for efficient cache management. Through this research, we provide new methodologies for design, modeling, and evaluation of effective caching mechanisms and resource management in various caching architectures of edge computing, CDNs, and CCN for ICN.

In Chapter 2, we first proposed an energy efficient design method to derive the optimal cache locations of content chunks in order to provide reference locations to evaluate energy efficiency for cache strategies, which can consider the tradeoff between the cache allocation power and traffic transmission power under the constraints of the caching hierarchy. Furthermore, we proposed a distributed cache mechanism to locally search for energy efficient cache locations of content chunks. In this mechanism, each *CN* pre-designs a fixed threshold of request rates of chunks for each caching hierarchy and judges whether or not to cache the chunks by comparing measured request rates with the threshold. In the simulation, we revealed the tradeoff between the cache allocation power and the traffic power for a chunk of content having different request rates and demonstrated that the proposed distributed caching is near to the optimal solution derived by the optimization model and can improve the total power consumption and the cache hit ratio in the target network compared with Pure LFU. Furthermore, we showed that the energy efficiency of the proposed method depends on the distribution of content popularity.

Second, we focus on constructing the approximation models of 2Q and ARC which can consider the interactions between separate queues for *one-timers* in Chapter 3. In the evaluations, we validated that the proposed models can provide a good approximation of the simulation results. As a result, the proposed models can easily analyze the statistical performance of 2Q and ARC at high accuracy. Finally, by using the model-based analysis, we confirmed that the cache performance of 2Q and ARC is better than that of LRU because the partition management of multiple queues can reduce the influence of *one-timers* on the cache performance of relatively popular objects.

Third, we focus on cache aging techniques based on *Time-To-Live* (TTL) of content in Chapter 4. The TTL-based caching facilitates analyzing cache characteristics and can realize appropriate resource management by setting efficient TTLs. Therefore, we proposed an analytical model to evaluate the cache characteristics of a distributed cache system. The proposed model is expressed by iterative matrix equations and can evaluate the impact of TTL-based caching on network resources and cache performance on multiplexed caching hierarchies. In the evaluations, we verified the validity of the proposed model and analyzed the impact on memory usage, power consumption, cache hit ratio, and average hop length when changing the TTL value of content.

In Chapter 5, as an enhanced mechanism of the TTL-based caching, we finally proposed an adaptive control mechanism which auto-tunes the TTL value of content based on predictive models to reduce the total resource cost in distributed cache systems and introduced a distributed solution. The proposed mechanism periodically decides the manipulated values of TTL by predicting the impact of the TTL values on the total resource cost in the hierarchical cache system. In the evaluations, we compared the control results estimated by the proposed model with those obtained by simulations. Furthermore, we analyzed the effectiveness of the proposed mechanism and showed that our mechanism can search for a TTL value of content to reduce the total resource cost by descending the slope of the cost function. Additionally, we analyzed the cache performance in the proposed control mechanism by using performance metrics such as cache miss ratio and average hop length and finally demonstrated the adaptability of the proposed mechanism.

In future management for ICN, it will be important to consider the tradeoff between the utilization efficiency of system resources in the distributed cache systems and service quality such as latency. Moreover, caching mechanisms which can efficiently and adaptively process nonstationary

accesses for content will be also required. Additionally, the scalability of the cache management in ICN should be improved in view of resolving addresses of content, disseminating content, and searching for target content in the distributed cache networks. We believe that the discussion in this thesis has useful implications for future research regarding these issues in content dissemination networks.

Bibliography

[1] S. Sivasubramanian, G. Pierre, M.V. Steen, G. Alonso, "Analysis of Caching and Replication Strategies for Web Applications," *IEEE Internet Computing*, vol. 11, no. 1, pp. 60–66, 2007.

[2] Y. Lin, B. Kemme, M.P. Martinez, R. Jimenez-Peris, "Enhancing Edge Computing with Database Replication," in *SRDS*. IEEE Computer Society, October 2007, pp. 45–54.

[3] G. Silvestre, S. Monnet, R. Krishnaswamy, P. Sens, "Caju: A Content Distribution System for Edge Networks," in *Euro-Par Workshops*, ser. Lecture Notes in Computer Science, vol. 7640. Springer, 2012, pp. 13–23.

[4] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, "Globally Distributed Content Delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, September 2002.

[5] A. Pathan, R. Buyya, "A Taxonomy and Survey of Content Delivery Networks," Melbourne, Australia, 2007.

[6] E. Nygren, R. K. Sitaraman, J. Sun, "The Akamai Network: A Platform for High-performance Internet Applications," *Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[7] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, B. Ohlman, "A Survey of Information-Centric Networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012.

[8] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, G. C. Polyzos, "A Survey of Information-Centric Networking Research," in *IEEE Communications Surveys & Tutorials*, vol. 19, no. 99, July 2013, pp. 1–26.

[9] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard, "Networking Named Content," in *Proceedings of the 5th ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT 2009)*, Rome, Italy, December 2009, pp. 1–12.

[10] W. Li, E. Chan, Y. Wang, D. Chen, S.Lu, "Cache Placement Optimization in Hierarchical Networks: Analysis and Performance Evaluation," in *7th International IFIP-TC6 Networking Conference*, Singapore, May 2008, pp. 385–396.

[11] H. Che, Y. Tung, Z. Wang, "Hierarchical Web Caching Systems: Modeling, Design and Experimental Results," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1305–1314, September 2002.

[12] S. Borst, V. Gupta, "Distributed Caching Algorithms for Content Distribution Networks," in *Proceedings of the 29th IEEE Conference on Computer Communications (INFOCOM 2010)*, San Diego, CA, USA, March 2010, pp. 1478–1486.

[13] T. Johnson, D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm." Morgan Kaufmann, 1994, pp. 439–450.

[14] N. Megiddo, D.S.Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache," in *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies*, ser. FAST '03, 2003, pp. 115–130.

[15] N. Megiddo, D. S. Modha, "Outperforming LRU with an Adaptive Replacement Cache Algorithm," *IEEE Computer*, vol. 37, no. 4, pp. 58–65, 2004.

[16] S. Imai, K. Leibnitz, M. Murata, "Energy Efficient Content Locations for In-Network Caching," in *Proceedings of APCC'12*, Jeju, Korea, October 2012, pp. 554–559.

[17] ——, "Energy-Aware Cache Management for Content-Centric Networking," in *Proceedings of First International WorkShop on Energy-Aware Systems, Communications and Security*, Barcelona, Spain, March 2013.

[18] ——, "Energy Efficient Data Caching for Content Dissemination Networks," *Journal of High Speed Networks*, vol. 19, no. 3, pp. 215–235, October 2013.

[19] Ministry of Economy, Trade, and Industry, "Green IT Initiative in Japan," 2008.

[20] M. Gupta, S. Singh, "Greening of Internet," in *Proceedings of ACM SIGCOMM'03*, Karlsruhe, Germany, August 2003, pp. 19–26.

[21] P. Mahadevan, P. Sharma, S. Banerjee, P. Ranganathan, "A Power Benchmarking Framework For Network Devices," in *Proceedings of NETWORKING'09*, vol. 5550, Aachen, Germany, May 2009, pp. 795–808.

[22] T. Harder, V. Hudlet, Y. Ou, D. Schall, "Energy Efficiency is not Enough, Energy Proportionality is Needed!" in *Proceedings of DASFAA'11*, Hong Kong, China, April 2011, pp. 226–239.

[23] A. Dan, D. Towsley, "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes," *SIGMETRICS Perform. Eval. Rev.*, vol. 18, no. 1, pp. 143–152, April 1990.

[24] M. Arlitt, C. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications," *IEEE/ACM Transaction on Networking*, vol. 5, pp. 631–645, October 1997.

[25] S. M. Abid, H. Youssef, "Impact of One-Timer/N-Timer Object Classification on the Performance of Web Cache Replacement Algorithms," in *Proceedings of Web Intelligence and Intelligent Agent Technology (WI-IAT)*, vol. 1, Toronto, ON, Canada, August 2010, pp. 208–211.

[26] P. Gill, M. Arlitt, Z. Li, "YouTube Traffic Characterization: a View from the Edge," in *Proceedings of IMC'07*, San Diego, CA, USA, October 2007, pp. 15–28.

[27] A. Belloum, L. O. Hertzberger, "Dealing with One-Timer-Documents in Web Caching," in *EUROMICRO*, vol. 2. IEEE Computer Society, August 1998, pp. 544–550.

[28] A. Mahanti, D. Eager, C. Williamson, "Temporal Locality and its Impact on Web Proxy Cache Performance," *PERFORMANCE EVALUATION*, vol. 42, pp. 187–203, 2000.

[29] S. M. Abid, H. Youssef, "Impact of One-Timer/N-Timer Object Classification on the Performance of Web Cache Replacement Algorithms," in *Web Intelligence*. IEEE, 2010, pp. 208–211.

[30] S. Imai, K. Leibnitz, M. Murata, "Modeling of Content Dissemination Networks on Multiplexed Caching Hierarchies," in *The Thirteenth International Conference on Networks*, Nice, Frace, February 2014, pp. 111–118.

[31] ——, "Adaptive TTL Control to Minimize Resource Cost in Hierarchical Caching Networks," *IEICE Transactions on Information and Systems*, March 2015.

[32] A. Fischer, J. F. Botero, M. T. Beck,H. Meer, X. Hesselbach, "Virtual Network Embedding: A Survey." *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[33] Y. Chen, J. Li, T. Wo,C. Hu, W. Liu, "Resilient Virtual Network Service Provision in Network Virtualization Environments." in *IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2010, pp. 51–58.

[34] C. Fricker, P. Robert, J. Roberts, "A Versatile and Accurate Approximation for LRU Cache Performance," in *Proceedings of the 24th International Teletraffic Congress (ITC)*, Krakow, Poland, September 2012, pp. 1–8.

[35] D. S. Berger, P. Gland, S. Singla, F. Ciucu, "Exact Analysis of TTL Cache Networks: The Case of Caching Policies driven by Stopping Times," *CoRR*, vol. abs/1402.5987, 2014.

[36] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, ser. Frontiers in Applied Mathematics. SIAM, 1995, no. 16.

[37] ——, *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics, 1999.

[38] U. Lee, I. Rimac, D. C. Kilper, V. Hilt, "Toward Energy-Efficient Content Dissemination," *IEEE Network*, vol. 25, no. 2, pp. 14–19, March 2011.

[39] U. Lee, I. Rimac, V. Hilt, "Greening the Internet with Content-Centric Networking," in *Proceedings of e-Energy*, Passau, Germany, October 2010, pp. 179–182.

[40] K. Guan, G. Atkinson, D. C. Kilper, "On the Energy Efficiency of Content Delivery Architectures," in *Proceedings of the 4th IEEE International Conference on Communications (ICC) Workshop on Green Communications*, Kyoto, Japan, June 2011.

[41] A. Schrijver, *Theory of Linear and Integer Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1986.

[42] C. Fricker, P. Robert, J. Roberts, "Impact of Traffic Mix on Caching Performance in a Content-Centric Network," in *Proceedings of IEEE NOMEN'12, Workshop on Emerging Design Choices in Name-Oriented Networking*, Orlando, Florida, USA, March 2012.

[43] D. Rossi, G. Rossini, "Caching Performance of Content Centric Networks under Multi-Path Routing," 2011.

[44] N. C. Fofack, D. Towsley, M. Badov, M. Dehghan, D. L. Goeckel, "An Approximate Analysis of Heterogeneous and General Cache Networks," RR-8516, INRIA, Tech. Rep., 2014. [Online]. Available: {https://hal.inria.fr/hal-00975339}

[45] M. Feldman, J. Chuang, "Service Differentiation in Web Caching and Content Distribution," in *Proceedings of IASTED International Conference on Communications and Computer Networks (CCN 2002)*, Cambridge MA, November 2002.

[46] M. Gallo, B. Kauffmann, L. Muscariello, A. Simonian, C. Tanguy, "Performance Evaluation of the Random Replacement Policy for Networks of Caches," *Performance Evaluation*, vol. 72, pp. 16–36, February 2014.

[47] E. J. Rosensweig, J. Kurose, D. Towsley, "Approximate Models for General Cache Networks," in *Proceedings of the 29th Conference on Information Communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, March 2010, pp. 1100–1108.

[48] N. C. Fofack, P. Nain, G. Neglia, D. Towsley, "Analysis of TTL-based Cache Networks," in *VALUETOOLS*. IEEE, 2012, pp. 1–10.

[49] ——, "Performance Evaluation of Hierarchical TTL-based Cache Networks," *Computer Networks*, vol. 65, pp. 212–231, March 2014.

[50] A. Tanaka, K. Tatsukawa, "Interreference Interval for Purged Objects: a New Metric for Design and Analysis of Web Caching Algorithms," in *Proceedings of IEEE Performance, Computing, and Communications Conference*, 2003, pp. 549–554.

[51] G. Carofiglio, V. Gehlen, D. Perino, "Experimental Evaluation of Memory Management in Content-Centric Networking," in *Proceedings of IEEE International Conference on Communications (ICC)*, Kyoto, Japan, June 2011.

[52] Y. T. Hou, J. Pan, B. Li, S. S. Panwar, "On Expiration-Based Hierarchical Caching Systems," *IEEE Journal on Selected Areas in Communications*, January 2004.

[53] I. Baev, R. Rajaraman,C. Swamy, "Approximation Algorithms for Data Placement in Arbitrary Networks," in *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Washington, DC, USA, January 2001, pp. 661–670.

[54] L. Qiu, V. N. Padmanabhan, G. M. Voelker, "On the Placement of Web Server Replicas," in *Proceedings of the 20th IEEE Conference on Computer Communications (INFOCOM 2001)*, Anchorage, AK, USA, April 2001, pp. 1587–1596.

[55] A. Leff, J. L. Wolf, "ReplicationAlgorithms in a Remote Caching Architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 11, pp. 1185–1204, November 1993.

[56] Z. Drezner, *Facility Location: A Survey of Applications and Methods*. Berlin, DEU: Springer, 1995.

[57] F. N. Santos, B. Ertl, C. Barakat, T. Spyropoulos, T. Turletti, "CEDO: Content-Centric Dissemination Algorithm for Delay-Tolerant Networks," in *MSWiM*. ACM, November 2013, pp. 377–386.