UNIVERSITY OF CALGARY

Markov Chain Analysis of Web Cache Systems

under TTL-based Consistency

by

Hazem Anwar Gomaa

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

April, 2013

© Hazem Anwar Gomaa 2013
Abstract

Caching plays a crucial role in improving Web-based services. The basic idea of Web caching is to satisfy the user’s requests from a nearby cache, rather than a faraway origin server (OS). This reduces the user’s perceived latency, the load on the OS, and the network bandwidth consumption.

This dissertation introduces a new analytical model for estimating the cache hit ratio as a function of time. The cache may not reach the steady state hit ratio (SSHR) when the number of Web objects, object popularity, and/or caching resources themselves are subject to change. Hence, the only way to quantify the hit ratio experienced by users is to calculate the instantaneous hit ratio (IHR). The proposed analysis considers a single cache with either infinite or finite capacity. For a cache with finite capacity, two replacement policies are considered: first-in-first-out (FIFO) and least recently used (LRU). In addition, the proposed analysis accounts for the use of two variants of the time-to-live (TTL) weak consistency mechanism. The first is the typical TTL (TTL-T), specified in the HTTP/1.1 protocol, where expired objects are refreshed using conditional GET requests. The second is TTL immediate ejection (TTL-IE) whereby objects are ejected as soon as they expire. Based on the insights from the analysis, a new replacement policy named frequency-based-FIFO (FB-FIFO) is proposed. The results show that FB-FIFO achieves a better IHR than FIFO and LRU.

Furthermore, the analytical model of a single cache is extended for characterizing the instantaneous average hit distance (IAHD) of two cooperative Web caching
systems, distributed and hierarchical. In these systems, the analysis considers fixed

caches that are always connected to the network, and temporary caches that ran-
domly join and leave the network. In the distributed cache system, the analysis

considers caches that cooperate via Internet cache protocol (ICP). In the hierarchical

cache system, the analysis accounts for two sharing protocols: leave copy everywhere

(LCE) and promote cached objects (PCO), which is a new protocol that is introduced

in this dissertation. The results show that PCO outperforms LCE in terms of the

IAHD, especially under TTL-T.
Acknowledgements

I am deeply indebted to my supervisor, Dr. Geoffrey Messier, for his guidance, support, and advice throughout my dissertation. Dr. Messier helped me to identify research problems and provided me with his full support in pursuing my ideas. Dr. Messier’s way of thinking and dealing with difficulties positively influenced not only my dissertation, but also my personal life.

I thank my co-supervisor, Dr. Robert Davies, for his ongoing discussions about my research and his assistance in providing funding to support my research.

I would like to express my sincere gratitude to Dr. Carey Williamson for his useful comments and crucial feedbacks that helped me a lot to improve the quality of my dissertation.

I thank TRLabs (Telecommunications Research Laboratories, Alberta) who provided financial support through their scholarship, which allowed me to pursue my degree.

I thank all my colleagues in FISH Lab for their ongoing help. I also would like to thank Martin F. Arlitt for providing technical materials that helped me during my research.

Finally, I deeply appreciate and give my heartfelt thanks to my parents, Fatma Mohamad and Anwar Gomaa, for their love and support, and my uncle, Dr. Ismail Gomaa, for encouraging me to continue my graduate studies.
To my lovely wife, Mayada Younes.

To my beautiful daughter, Jumana Gomaa.
# Table of Contents

Abstract .................................................................................................................. ii  
Acknowledgements ................................................................................................. iv  
Dedication .................................................................................................................. v  
Table of Contents ...................................................................................................... vi  
List of Tables ............................................................................................................ ix  
List of Figures and Illustrations ................................................................................ x  
List of Symbols, Abbreviations, Nomenclature ........................................................ xiii  
1 Introduction ............................................................................................................ 1  
  1.1 Proposed Web Cache Systems ................................................................. 3  
    1.1.1 Single Cache System (SCS) ............................................................. 3  
    1.1.2 Hierarchical Cache System (HCS) ................................................ 6  
    1.1.3 Distributed Cache System (DCS) .................................................... 7  
    1.1.4 Temporary Cache (TC) ................................................................. 9  
1.2 Evaluation Methodology .................................................................................... 13  
1.3 Motivations and List of Contributions ............................................................ 18  
1.4 Dissertation Roadmap ...................................................................................... 20  
2 Background ............................................................................................................. 21  
  2.1 HTTP and Cache Consistency Mechanisms ................................................. 21  
  2.2 Cache Replacement Policies ........................................................................ 26  
  2.3 Access Pattern Characteristics ..................................................................... 28  
  2.4 SCS Analytical Models in the Literature ...................................................... 30  
  2.5 Research on Hierarchical Cache Systems ................................................... 32  
    2.5.1 HCS Sharing Protocols ..................................................................... 33  
    2.5.2 HCS Analytical Models in the Literature ........................................ 35  
  2.6 Research on Distributed Caching Systems ................................................... 36  
    2.6.1 DCS Sharing Protocols ..................................................................... 36  
    2.6.2 DCS Analytical Models in the Literature ........................................ 38  
  2.7 Research on Mobile Cache Systems .............................................................. 39  
3 Analysis of the Single Cache System (SCS) .......................................................... 43  
  3.1 Analysis Assumptions ...................................................................................... 44  
  3.2 Analysis of Infinite Cache .............................................................................. 46  
    3.2.1 Infinite Cache under NEM .................................................................. 47  
    3.2.2 Infinite Cache under TTL-IE .......................................................... 48  
    3.2.3 Infinite Cache under TTL-T ............................................................ 50  
  3.3 Analysis of Perfect-LFU ................................................................................ 52  
  3.4 Analysis of FIFO ......................................................................................... 53  
    3.4.1 FIFO Cache under NEM .................................................................... 54  
    3.4.2 FIFO Cache under TTL-IE .............................................................. 57  
    3.4.3 FIFO Cache under TTL-T ............................................................... 59  
  3.5 Analysis of LRU ......................................................................................... 61  
    3.5.1 LRU Cache under NEM ................................................................. 61  
    3.5.2 LRU Cache under TTL-IE ............................................................... 62
List of Tables

3.1 Iterative analysis for estimating the IHR of a single FIFO cache under NEM. .......................................................... 56
3.2 SCS: IHR evaluation factors and levels. .............................. 74
3.3 The IHR of FIFO, where $\mu_e = 0$ and $\beta = 500$. ...................... 77
3.4 SCS: SSHR evaluation factors and levels. ............................... 84

4.1 Iterative analysis for estimating the IHR of a TLC running LRU under TTL-IE. .......................................................... 103
4.2 HCS: evaluation factors and levels. ..................................... 105

5.1 DCS: evaluation factors and levels. ..................................... 124
5.2 SSAHD of ICP and ICP-COCO under NEM and TTL-IE, where $\mu_e = 1$, assuming FCs. ....................................................... 131
5.3 SSAHD of ICP and ICP-COCO under NEM and TTL-IE, where $\mu_e = 1$, assuming TCs. ....................................................... 131

A.1 The contraction mapping of $\cos(x)$ [33]. .............................. 144
A.2 FIFO iterative analysis, where $M = 1000$ and $C = 100$. ........... 149

B.1 List of new symbols used in Appendix B. .............................. 160
B.2 SSHR of FIFO root cache, where $C_R = C_L$, $M = 1000$, and $\alpha = 0.8$. 167
# List of Figures and Illustrations

1.1 Network topology for a single cache. ........................................ 2
1.2 Network topology for a 2-level cache hierarchy, with a user’s browser cache at the first level and an ISP forward cache at the second level. 2
1.3 Network topology for the proposed 2-level HCS, with many leaf caches at the first level and one root cache at the second level. .......... 6
1.4 Network topology for the proposed DCS. .................................. 8
1.5 Temporary caching in MANETs. ............................................... 10
1.6 Network topology for temporary caches in the HCS. ..................... 11
1.7 Network topology for temporary caches in the DCS. ....................... 12
1.8 A sample function of a continuous-time stochastic process $X(t)$ whose state space is \{0, 1\}. .............................................. 15
1.9 Calculating the IHR by finding the ensemble average of $N$ experiments. 16
1.10 Time intervals where either IHR or SSHR should be used to evaluate cache performance. ..................................................... 17
2.1 Sample HTTP request and response headers. ................................. 23
2.2 Object access under TTL-T and TTL-IE. .................................... 25
3.1 Infinite Cache Markov chain for object $i$ under NEM. ..................... 47
3.2 Infinite Cache Markov chain for object $i$ under TTL-IE. ................. 49
3.3 Infinite Cache Markov chain for object $i$ under TTL-T. ................. 50
3.4 FIFO Markov chain for object $i$ under NEM. ................................ 54
3.5 FIFO Markov chain for object $i$ under TTL-IE. ............................. 58
3.6 FIFO Markov chain for object $i$ under TTL-T. ............................. 60
3.7 LRU Markov chain for object $i$ under NEM. ................................ 62
3.8 LRU Markov chain for object $i$ under TTL-IE. ............................. 63
3.9 LRU Markov chain for object $i$ under TTL-T. ............................. 64
3.10 Calculating $S(t_1, i)$, assuming stationary access pattern within the interval $(t_1, t_1 - \Delta t)$. ............................................. 66
3.11 FB-FIFO: a new object enters the cache. .................................... 69
3.12 FB-FIFO: an object in $S_u$ is accessed. .................................... 70
3.13 FB-FIFO flow chart. .......................................................... 71
3.14 Simulated IHR is calculated at every simulation timestep, $\delta t$. .... 73
3.15 IHR, where $C = 500$, $\mu_e = 0$, and $\beta = 500$. ...................... 75
3.16 IHR, where $\mu = 0$ and $\beta = 500$. ...................................... 77
3.17 IHR, where $C = 300$ and $\beta = 500$. ................................... 79
3.18 The relative change in the IHR when $\mu_e$ increases from 0.5 to 1, where $C = 300$ and $\beta = 500$. ........................................... 80
3.19 IHR, where $C = 300$, $\mu_e = 0.5$, and $\beta = 250$. .................... 80
3.20 IHR, where $C = 100$ and $\beta = 200$. .................................... 81
3.21 IHR, where $\mu_e = 0$ and $\beta = 200$. .................................... 83
3.22 SSHR of FIFO. ............................................................... 85
3.23 SSHR of LRU ................................................................. 86
3.24 SSHR of FB-FIFO ........................................................... 87

4.1 An example of calculating the IHR of the root cache, while the leaf caches are in a transient period. ......................... 95
4.2 LRU Markov chain for object i in PCO root cache under TTL-IE. ................................................................. 98
4.3 The proposed TLCs network model ....................................... 100
4.4 Birth-death Markov chain model of the TLCs population ........ 101
4.5 SSHR (in percent) and SSAHD of LCE, LCD, and PCO under TTL-IE. ............................................... 107
4.6 SSHR (in percent) and SSAHD of LCE, LCD, and PCO under TTL-T, where $\mu_e = 1$ ................................................. 109
4.7 IHR (in percent) and IAHD of LCE, LCD, and PCO, where $C_R = 50$ and $\mu_e = 0$. .................................................. 111
4.8 IHR (in percent) and IAHD of LCE, LCD, and PCO, where $C_R = 250$ and $\mu_e = 0$ .................................................. 112
4.9 SSHR (in percent) of LCE, LCD, and PCO under TTL-IE and with TLCs ................................................................. 114

5.1 Network topology for temporary caches in the DCS. .............. 121
5.2 The proposed distributed TCs network model ......................... 122
5.3 The proposed DCS: ICP vs. ICP-COCO ................................. 123
5.4 SSHR (in percent) and SSAHD of the DCS with FCs, using ICP under TTL-IE or TTL-T, where $\mu_e = 1$. ......................... 125
5.5 SSHR (in percent) and SSAHD of the DCS with TCs, using ICP under TTL-IE, where $\mu_e = 1$. ................................. 125
5.6 SSHR (in percent) and SSAHD of the DCS with FCs, using ICP under TTL-IE or TTL-T, where $\mu_e = 10$. ...................... 126
5.7 SSHR (in percent) and SSAHD of the DCS with FCs, using ICP-COCO under TTL-IE or TTL-T, where $\mu_e = 1$. ................... 128
5.8 SSHR (in percent) and SSAHD of the DCS with TCs, using ICP-COCO under TTL-IE, where $\mu_e = 1$. ....................... 129
5.9 SSHR (in percent) and SSAHD of the DCS with FCs, using ICP-COCO under TTL-IE or TTL-T, where $\mu_e = 10$. .................. 130

6.1 LRU Markov chain for object i under TTL-IE, assuming FCs and dependent-LRU ................................................................. 140
6.2 LRU Markov chain for object i under TTL-T, assuming FCs and dependent-LRU ................................................................. 141

A.1 $\cos(x)$ is a contraction mapping [33] ...................................... 143
A.2 $e^x$ is a contraction mapping [33] ........................................ 145
A.3 FIFO iterative analysis, where $M = 100$ and $C = 50$. ............... 148
A.4 FIFO iterative analysis, where $M = 1000$ and $C = 100$. .......... 149

B.1 Case 2: $K = 1$ and $C_R > C_L$, where there is a probability that the leaf cache has an object that does not exist in the root cache. .......... 163
B.2 SSHR (in percent) of FIFO root cache, where $K = 1$ and $\alpha = 0.8$. . . 164
B.3 SSHR (in percent) of FIFO root cache, where $K = 4$, $C_L = 20$, $M = 1000$, and $\alpha = 0.8$. . . . . . . . . . . . . . . . . . . . . . . . . . . . 168
B.4 SSHR (in percent) of FIFO root cache, assuming $K_g = C_R/C_L$, where $K = 4$, $C_L = 20$, $M = 1000$, and $\alpha = 0.8$. . . . . . . . . . . . . . . . . . . . . . . . . . . . 169
B.5 SSHR (in percent) of LRU root cache, where $K = 4$, $C_L = 20$, $M = 1000$, and $\alpha = 0.8$. . . . . . . . . . . . . . . . . . . . . . . . . . . . 169
# List of Symbols, Abbreviations, Nomenclature

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICH</td>
<td>One-Copy Heuristic</td>
</tr>
<tr>
<td>CARP</td>
<td>Cache Array Routing Protocol</td>
</tr>
<tr>
<td>CFS</td>
<td>Cache-Fresh State</td>
</tr>
<tr>
<td>CES</td>
<td>Cache-Expired State</td>
</tr>
<tr>
<td>DCS</td>
<td>Distributed Cache System</td>
</tr>
<tr>
<td>FIFO</td>
<td>First-In-First-Out</td>
</tr>
<tr>
<td>FB-FIFO</td>
<td>Frequency-Based-FIFO</td>
</tr>
<tr>
<td>FC</td>
<td>Fixed Cache</td>
</tr>
<tr>
<td>FLC</td>
<td>Fixed Leaf Cache</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>HCS</td>
<td>Hierarchical Cache System</td>
</tr>
<tr>
<td>Infinite Cache</td>
<td>Cache with Infinite Capacity</td>
</tr>
<tr>
<td>IHR</td>
<td>Instantaneous Hit Ratio</td>
</tr>
<tr>
<td>ICP</td>
<td>Internet Cache Protocol</td>
</tr>
<tr>
<td>ICP-COCO</td>
<td>ICP-Cache Origin Copy Only</td>
</tr>
<tr>
<td>IAHD</td>
<td>Instantaneous Average Hit Distance</td>
</tr>
<tr>
<td>IRM</td>
<td>Independent Reference Model</td>
</tr>
<tr>
<td>LRU</td>
<td>Least Recently Used</td>
</tr>
<tr>
<td>LCE</td>
<td>Leave Copy Everywhere</td>
</tr>
<tr>
<td>LCD</td>
<td>Leave Copy Down</td>
</tr>
<tr>
<td>MANET</td>
<td>Mobile Ad-hoc Network</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>NCS</td>
<td>Non-Cache State</td>
</tr>
<tr>
<td>NEM</td>
<td>Non-Expiry Model</td>
</tr>
<tr>
<td>OS</td>
<td>Origin Server</td>
</tr>
<tr>
<td>PCO</td>
<td>Promote Cached Objects</td>
</tr>
<tr>
<td>Perfect-LFU</td>
<td>Perfect Least Frequently Used</td>
</tr>
<tr>
<td>SSHR</td>
<td>Steady State Hit Ratio</td>
</tr>
<tr>
<td>SCS</td>
<td>Single Cache System</td>
</tr>
<tr>
<td>SSAHD</td>
<td>Steady State Average Hit Distance</td>
</tr>
<tr>
<td>TC</td>
<td>Temporary Cache</td>
</tr>
<tr>
<td>TLC</td>
<td>Temporary Leaf Cache</td>
</tr>
<tr>
<td>TTL</td>
<td>Time-to-Live</td>
</tr>
<tr>
<td>TTL-IE</td>
<td>TTL Immediate Ejection</td>
</tr>
<tr>
<td>TTL-T</td>
<td>Typical TTL</td>
</tr>
<tr>
<td>VANET</td>
<td>Vehicular Ad-hoc Network</td>
</tr>
<tr>
<td>Web</td>
<td>World Wide Web</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The exponential growth rate of the World Wide Web (Web)\(^1\) has led to a dramatic increase in Internet traffic over the past decade. Hence, it has become challenging to provide responsive\(^2\) without investing in costly network links and utilities \([1,2,3,4,5]\) Web-based services.

In Web applications, the user’s perceived latency can be reduced by serving the requested Web objects to the user from a closer cache (a cache hit), rather than from the OS (a cache miss), as shown in Fig. 1.1. Moreover, caching also plays a crucial role in reducing redundant traffic between users and the OSs. This improves Web-based services by reducing the load on the OS, and the network bandwidth consumption. On the other hand, cache misses increase the user’s perceived latency due to the additional time that a cache requires to resolve the requests \([6,7,8,9,10,11,12,13,14,15]\). Caching is critical not only in network applications but also in microprocessor systems. Very fast caches are usually used to hide the speed gap between main memory and the CPU \([16,17,18,19,20,21,22,23]\).

Web caching can occur at various locations in the Internet hierarchy, for example, the user’s Web browser, the institution’s Internet service provider (ISP), the regional Internet hub, and/or the national Internet hub backbone. Several caches along the

---

1World Wide Web (Web) is a distributed system that allows users to retrieve Web objects (e.g. text, image, audio file, video files) over the Internet via Hypertext Transfer Protocol (HTTP).
2Users browsing the Internet feel that responses are “instant” when latency is less than 100-200 ms. [5].
path between the user and the OS may cooperate in order to further improve the
user’s perceived latency [24, 25, 26, 27, 28, 29, 30]. For example, the user’s browser
cache can resolve a request from a cache located at the ISP, without contacting the
OS, as shown in Fig. 1.2. However, successive cache misses at the cooperative caches
will degrade the user’s perceived latency. Moreover, caches that are closer to the
OS handles traffic generated from more users, and they may become bottlenecks and
cause more delays [3, 7, 31]. Hence, a careful design of cooperative caching schemes is
crucial.

Figure 1.2: Network topology for a 2-level cache hierarchy, with a user’s browser
cache at the first level and an ISP forward cache at the second level.

In this dissertation, a novel Markov chain analytical model for characterizing the
performance of Web cache systems is proposed. The proposed analytical model is
based on the contraction mapping theory [32, 33, 34]. Furthermore, based on the in-
sights from the proposed analytical model, this dissertation introduces a new replace-
ment policy, named frequency-based-FIFO (FB-FIFO) that improves the performance of a single cache. Moreover, a new sharing protocol, called promote cached objects (PCO), is proposed in order to enhance the performance of a cache hierarchy, as will be discussed in detail in the following chapters.

The remainder of this chapter is organized as follows. First, the basics of the proposed cache systems are discussed in Section 1.1. Then, the evaluation methodology is discussed in Section 1.2. Following in Section 1.3 are the motivations and the list of contributions. Finally, the outline of this dissertation is presented in Section 1.4.

1.1 Proposed Web Cache Systems

In this dissertation, three Web cache systems are considered: a single cache system (SCS), a hierarchical cache system (HCS), and a distributed cache system (DCS). These systems are described briefly below in Sections 1.1.1, 1.1.2, and 1.1.3, respectively.

1.1.1 Single Cache System (SCS)

In the proposed SCS, the caching activity of a single Web cache is considered. The single cache can be installed on a proxy server, as shown in Fig. 1.1, or hosted by a Web user (e.g. a browser cache).

In the SCS, the HTTP/1.1 protocol\textsuperscript{3} allows communication between the user, the OS, and the intermediate cache. The user’s requests are directed to the cache. If

\footnotesize\textsuperscript{3}The current version of the HTTP protocol is known as HTTP/1.1. The earlier versions of HTTP include HTTP/0.9 and HTTP/1.0.
the cache has the requested object, the user downloads the object from the cache. In this case, the request is counted as a cache hit. Otherwise, the cache downloads the requested object from the OS and stores a copy of the object to satisfy future requests. While downloading the object, the cache forwards the object to the requesting user. In this scenario, the request is counted as a cache miss. Note that the HTTP/1.1 response may prevent the cache from retaining a copy of the requested object using the Cache-Control header [1]. However, throughout this dissertation, it is assumed that all the requests are for cacheable objects.

An important factor that affects cache performance is the cache consistency mechanism. If the cache fails to regularly validate the cached objects (i.e. update them to be identical to the ones at the OS), then it may serve an invalid object to users. Hence, cache consistency must be maintained to ensure (or at least increase the probability) that the cache serves valid objects [35, 36, 37, 38, 39].

HTTP/1.1 allows the OS to specify an explicit expiration time for the requested object, using either the Expires header, or the max-age directive of the Cache-Control header. The difference between the time when the object will expire (becomes stale) and the current time is called the Time-To-Live (TTL) value. The object is said to be fresh as long as its TTL value is positive. The TTL value decreases linearly with time and the object expires when its TTL value reaches zero. The cache may serve the requests for fresh objects without contacting the OS (i.e. the cache assumes that the cached object is valid as long as it is fresh). Note that it is possible that a fresh object becomes invalid, especially when the OS generates optimistic estimations for the TTL values. Hence, using the TTL mechanism provides weak cache consistency,
as will be discussed in detail in Chapter 2.

In the typical implementation of the TTL weak consistency in HTTP/1.1 (TTL-T) [1], if an expired object is requested from the cache, the cache has to contact the OS in order to validate this object (i.e. check if it is identical to the one on the OS). This will be referred to as a validation request. When the OS receives a validation request, it sends the updated object with a new TTL value to the cache only if the expired object is invalid. Another implementation of the TTL weak consistency was used in the Harvest and Squid caches [28,29,39]. In this implementation, the cached objects are ejected once they expire. This implementation will be referred to as TTL immediate ejection (TTL-IE). In this dissertation, both TTL-T and TTL-IE consistency mechanisms are considered in the proposed analysis. Moreover, a special case where the objects do not expire is also considered. This will be called the non-expiry model (NEM).

In this dissertation, the analysis for a single cache with infinite or finite size is introduced in Chapter 3. If the cache has an infinite size, then it can store all of the requested objects. Otherwise, for a cache with finite capacity, a replacement policy is applied to make room for the requested object, if the cache is full. Three traditional replacement policies are considered:

- Perfect-LFU: ejects the least frequently used object from the cache, and keeps a record of the number of requests for each object, even after the object is ejected.

- First-In-First-Out (FIFO): ejects the object that was brought into the cache earliest.

- Least Recently Used (LRU): ejects the least recently requested object.
Moreover, a new replacement policy, called frequency-based-FIFO (FB-FIFO), is proposed. FB-FIFO outperforms FIFO by creating a variable-sized protected cache segment for objects that are not effectively one-timers. Note that a one-timer is an object that is requested once, and will never be requested again. Furthermore, the object is effectively a one-timer if its request rate is too low to generate a cache hit [25], as will be discussed further in Chapter 3.

1.1.2 Hierarchical Cache System (HCS)

In the proposed HCS, the caching activity of a two-level hierarchical cache system is considered. The caches at the first level are called leaf caches. Each group of users is associated with a leaf cache. The leaf caches do not cooperate with each other. Furthermore, the second level has one root cache, as shown in Fig. 1.3. The HCS satisfies the user’s request from the leaf cache, if the leaf cache has the requested object. Otherwise, the leaf cache resolves the request from the root cache, if the root cache has the requested object. If the request cannot be satisfied from the leaf cache or the root cache, then the request will be sent to the OS.

Figure 1.3: Network topology for the proposed 2-level HCS, with many leaf caches at the first level and one root cache at the second level.
Cooperation between caches at different network levels requires a sharing protocol [24, 40, 41, 42]. The purpose of a HCS sharing protocol is to decide which cache in the HCS should store a copy of a requested object. Hence, the sharing protocol can be seen as an admission mechanism. Note that the replacement decision (i.e. deciding which object to eject from the cache to make room for a new object), is made independently at each cache in the cache hierarchy.

In this dissertation, two traditional sharing protocols are considered: leave copy everywhere (LCE) and leave copy down (LCD). In LCE, when the object is found, either at a cache or at the OS, it travels down the hierarchy and a copy is left at each of the intermediate caches. In LCD, when the object is found, it travels down the hierarchy and only one copy is left at the cache situated just below the level where the object is found. Furthermore, the design and analysis for a new hierarchical sharing protocol, called promote cached objects (PCO), are introduced in this dissertation. PCO improves the user's perceived performance by preserving the root cache for objects that are not effectively one-timers. In Chapter 4, the analysis for the HCS using LCE or PCO is presented.

1.1.3 Distributed Cache System (DCS)

In the proposed DCS, cooperation between caches that belong to the same network level (sibling caches) is considered. The goal of the DCS is to resolve the cache misses using a nearby sibling caches, rather than a faraway OS. Internet Cache Protocol (ICP) was designed in the Harvest project [28, 29, 39] to allow object retrieval from sibling caches, as shown in Fig. 1.4. If the user's request is satisfied by the associated
cache (local cache), then it will be counted as a *local hit*. Otherwise, if the user’s request is satisfied by any other cache in the DCS (remote caches), then it will be counted as a *remote hit*.

Figure 1.4: Network topology for the proposed DCS.

In ICP, the cache distinguishes between local requests (by associated users) and remote requests (i.e. requests directed from sibling caches). For example, a cache in the DCS does not store an object based on a remote request [29, 43]. This is because, in ICP, the cache does not resolve a remote request if it does not have the requested object. Hence, the remote cache only resolves cache hits. Moreover, each cache in the DCS manages its content and runs its replacement policy independently (i.e. caches in the DCS do not coordinate their replacement policies). For example, the local LRU cache list is not updated because of remote requests. Also, for Perfect-LFU, the popularity of objects as seen by a cache is not updated because of remote requests, and so on [44].

Note that ICP allows multiple copies of the same object to be stored in different
locations in the DCS. However, it might be more advantageous to use the available capacities in the DCS whereby only one copy of each object can be stored in the DCS at a time, as suggested by the one-copy heuristic (1CH) [45].

In Chapter 5, the analysis for the DCS is presented. The proposed analysis adopts the same sharing rules specified by ICP. Moreover, a modified version of ICP, named ICP-cache origin copy only (ICP-COCO), is evaluated using simulations. Under NEM and TTL-IE, ICP-COCO follows 1CH, where the object that is served by a remote cache is not locally cached. This will be discussed in detail in Chapter 5.

1.1.4 Temporary Cache (TC)

So far, it is assumed that the leaf cache in the HCS, or the cache in the DCS, is a fixed cache (FC). In this dissertation, the cache is called fixed if it is installed on a proxy server that is always connected to the network and available to serve the users’ requests. Furthermore, this dissertation also considers the case where a cache is available only for a short time. This will be referred to as the temporary cache (TC).

A TC can be hosted by a wireless mobile device, such as a laptop, a smart phone, or a vehicle that connects to a wireless infrastructure, such as WLAN or a cellular network, in order to access Web objects over the Internet. While connected, a TC may serve other users in the WLAN using peer-to-peer (P2P) communication. Fig. 1.5 shows a Mobile Ad-hoc Network (MANET) where mobile devices share information using P2P communication, without using fixed infrastructure. In Fig. 1.5, it is assumed that some mobile devices have extra cache storage and power to host TCs.
Moreover, low-power mobile devices, which do not host TCs, relay their requests to the mobile devices with TCs, which are assumed to be located in a better coverage area (i.e. mobile devices with TCs are closer to the access point).

Therefore, temporary caching can be used to reduce the amount of traffic experienced by an access point with limited bandwidth. Furthermore, temporary caching accelerates access to the required objects and saves the battery of an individual low-power mobile device that is served by a nearby TC instead of a faraway congested access point [13,46,47,48,49,50]. On the other hand, once a TC disconnects from the network, the cache system loses the objects that were cached on this TC. Furthermore, the advantages of mobile caching have to be balanced with the increased energy consumption of the mobile devices participating in the caching system [9]. This dissertation explores the performance of the proposed HCS and DCS in the presence of...
For the HCS, it is assumed that multiple MANETs are served by a fixed root cache. Moreover, each MANET contains a single TC that acts as a leaf cache. This will be called a temporary leaf cache (TLC). For example, consider some vehicles that serve as TLCs in a cache hierarchy with a root cache installed at the 3G Universal Mobile Telecommunication System (UMTS), as shown in Fig. 1.6. In this case, a Vehicular Ad Hoc Network (VANET) \([36, 51, 52, 53, 54, 55, 56, 57]\) maintains the communication between each TLC and its associated vehicles. While on the road, vehicles browse the Internet and initiate Web requests for news, local traffic information, and local map information. In a VANET, the requests are directed to the associated TLC (closest TLC to the user). If the request cannot be satisfied from the associated TLC, then the

---

\(^4\)The study for the trade-off between the energy consumption of mobile devices, due to participating in a cooperative cache system, and the performance of the cache system has been published in [9]. However, this issue is beyond the scope of this dissertation.
request is sent to the OS through a macrocell base station. In this example, a TLC plays the same role as a picocell base station in the sense that both reduce the load on the macrocell base station and enhance network capacity and coverage\textsuperscript{5} [58, 59, 60].

![Network topology for temporary caches in the DCS.](image)

Figure 1.7: Network topology for temporary caches in the DCS.

For the DCS, a single MANET with many TCs is considered. In the previous VANET example, when connected to the VANET, some vehicles within the same VANET may form a DCS to serve other vehicles, as shown in Fig. 1.7. In the proposed DCS, the user’s request is satisfied from the local TC (closest TC to the user), if the local TC has the requested object. Otherwise, the local TC will query the other TCs (remote TCs) in the DCS for that object. If the object cannot be found in any TC in the DCS, the local TC downloads the requested object from the OS via an infrastructure access point, and forwards the object to the requesting user.

\textsuperscript{5}Frequency planning is beyond the scope of this dissertation.
The analysis of the proposed HCS and DCS with temporary caches is discussed in detail in Chapter 4 and Chapter 5, respectively.

1.2 Evaluation Methodology

Evaluating the performance of cache systems accurately is crucial in network planning. Particularly, it is important to estimate the actual costs and benefits of applying a cache scheme prior to installation. Cache performance is commonly evaluated using analytical models [7, 8, 11, 12, 40, 41, 61, 62, 63, 64, 65, 66], and/or simulations [9, 14, 67, 68, 69].

There are two common approaches in cache simulations: trace-based and analytical-based. The trace-based approach uses empirical traces. On the other hand, the analytical-based approach uses mathematical models for the workload characteristics of interest, and uses random number generation to produce synthetic workloads that statistically conform to the adopted mathematical models. In comparing the two approaches, the analytical-based approach provides more flexibility in generating customized workloads, which incorporate selected characteristics, such as object popularity and temporal locality [14].

In this dissertation, the proposed cache systems are evaluated using a Markov chain analysis and analytical-based simulations. A synthetic workload generator and an event-driven simulator were developed using C++ to validate the analytical results and to provide the flexibility required to evaluate caching policies and protocols that are not considered in the proposed analysis.
The performance of a cache system is usually evaluated using the hit ratio, the byte hit ratio, and the average hit distance metrics \[14, 41, 70\]. The hit ratio is the fraction of the requests that are served from the cache instead of the OS. This is equivalent to the probability of finding the requested object in cache. Similarly, the byte hit ratio is the percentage of bytes that are served from the cache. Assuming that all objects have the same size, the byte hit ratio is the object size multiplied by the hit ratio. The average hit distance is the average number of links traversed to obtain the requested object. In this dissertation, the hit ratio and the average hit distance will be used as evaluation metrics\(^6\).

The performance of a cache system changes as a function of time when the cache resources (e.g. temporary caches), or the access pattern, are subject to change \[4, 71\]. In this dissertation, two types of access patterns are considered:

- A *stationary access pattern*: is when the characteristics of the access pattern do not change with time (i.e. fixed request rate, object popularity, etc.).

- A *non-stationary access pattern*: occurs when the characteristics of the access pattern exhibit variations (e.g. due to the generation of new popular objects at the OS, such as news headlines and new videos).

Since the Web users join the network for a limited duration, the user’s perceived latency varies according to the instantaneous performance of the cache. Therefore, in this dissertation, the cache hit ratio is evaluated as a function of time \[4, 71\]. This will be referred to as the *instantaneous hit ratio* (IHR), which is the ratio between

\(^6\)The proposed analysis assumes that all objects have the same size, as will be discussed in Chapter 3. Hence, the byte hit ratio is not considered.
the total number of requests that are satisfied from the cache and the total number of requests initiated at a certain time.

It is important to note that the IHR does not represent an average that is calculated across a particular time interval\(^7\). Rather, the IHR is equivalent to the probability of finding the requested object in the cache at a particular instant in time. For example, consider an experiment of accessing one object from the cache at time \(t\). The output of this experiment is a Bernoulli random variable, \(X(t)\), where \(X(t) = 1\) if the object is found in cache (i.e. cache hit), and \(X(t) = 0\) if the object is not found (i.e. cache miss). Hence, \(\{X(t), t \geq 0\}\) is a discrete-state, continuous-time stochastic process where the state space of \(X(t)\) is \(\{0, 1\}\). Fig. 1.8 shows a sample function (realization) of \(X(t)\).

Figure 1.8: A sample function of a continuous-time stochastic process \(X(t)\) whose state space is \(\{0, 1\}\).

In this experiment, the IHR at time \(t\), \(H(t)\), is equivalent to the probability that \(X(t) = 1\). Hence, \(H(t)\) is the ensemble average of repeating the experiment \(N\) times (at the same time \(t\)), as shown in Fig. 1.9. In Fig. 1.9, \(H(t)\) is calculated by counting the number of hits, \(N_H(t)\) (i.e. the number of times that the event \(X(t) = 1\) occurs),

\(^7\) The average hit ratio over an interval \([0, T]\) denotes the fraction of requests that are satisfied from the cache instead of the OS within this interval.
such that $H(t) = N_H(t)/N$ where $N$ is sufficiently large to attain a certain level of confidence \[72\].

Figure 1.9: Calculating the IHR by finding the ensemble average of $N$ experiments.

Assuming a stationary access pattern, an empty cache at time $t = 0$ eventually converges to a steady state hit ratio (SSHR) at time $t_s$. In this case, the IHR is evaluated within the interval $(0, t_s)$, as shown in Fig. 1.10(a). Assuming a non-stationary access pattern where the access pattern changes at time $t_v$, the cache that operates in the steady state since $t_{s_1}$ enters a transient period where the IHR converges to a new SXHR at time $t_{s_2}$, as shown in Fig. 1.10(b). Moreover, the cache may not reach a steady state due to the frequent changes in the access pattern. In this
scenario, the cache keeps switching from one transient period to another, as shown in Fig. 1.10(c).

Figure 1.10: Time intervals where either IHR or SSHR should be used to evaluate cache performance.

During the transient periods (e.g. \((0, t_{s1})\) and \((t_v, t_{s2})\) in Figs 1.10(a) and 1.10(b)), calculating the IHR is the only way to evaluate the user’s perceived latency. Note that the main goal of the proposed cache systems is to improve the user’s perceived latency, which is the duration between the time when the user initiates the request and the time when the response is received. For the SCS, a higher IHR implies that users experience a smaller latency.

Furthermore, for the HCS and the DCS, the user’s perceived latency varies according to the location where the requested object is found. Hence, the user’s perceived
latency at a certain time can only be characterized by evaluating the average number of links traversed to obtain the requested object at this time. This is called the instantaneous average hit distance (IAHD). A lower IAHD implies that users experience a smaller latency.

1.3 Motivations and List of Contributions

The proposed analytical model in this dissertation provides solutions for two fundamental problems: (1) estimating the instantaneous cache performance, assuming fixed cache (FC) or temporary cache (TC); and (2) incorporating the TTL-based consistency mechanism in the cache analysis.

Studies in [4, 73, 74, 75, 76] have shown that the access patterns obtained from real traces exhibit strong variations due to many factors, such as a growing number of objects, change in objects popularity, or a flash crowd\(^8\). Therefore, since the Web user joins the network for a limited duration, the only way to calculate the hit ratio experienced by the user is to estimate the IHR and the IAHD.

Moreover, estimating the IHR and the IAHD is crucial when the cache resources are subject to change. For example, in the DCS with TCs, the total cache size available for the cache system changes with time. Also, the TC is connected to the network for a short period. Hence, it is important for this TC to reach a better hit ratio quickly in order to enhance the hit ratio experienced by more users [12, 36, 71]. For example, in Fig. 1.10(a), the users who join the network while the TC is in the

\(^8\)A large spike or surge in traffic to a particular Web site.
beginning of the transient state will experience low hit ratios. On the other hand, users who join the network while the TC is closer to the steady state will experience a better hit ratio.

However, the analytical models in the literature focused on evaluating the cache performance in the steady state, assuming fixed cache resources, but did not provide information about the cache performance during transient periods.

The proposed analysis considers NEM, TTL-IE, and TTL-T consistency mechanism. TTL-T is the most widely used cache consistency mechanism on the Internet today \[28,35,37,39,77\]. However, for simplification sake, the analytical models in the literature considered either TTL-IE or NEM. More background regarding the related analytical models in the literature will be discussed in Chapter 2.

Furthermore, based on the insights from the proposed analytical model, this dissertation introduces a new replacement policy, named frequency-based-FIFO (FB-FIFO), and a new sharing protocol, called promote cached objects (PCO), in order to improve the user’s perceived latency. The performances of FB-FIFO and PCO are evaluated under NEM, TTL-IE, and TTL-T, as will be discussed in detail in the following chapters.

The main contributions of this dissertation are summarized in the following:

1- Introduction of a new analytical model for estimating the IHR of the SCS. The cache may have an infinite size, or a finite size. For a finite-sized cache, FIFO and LRU are considered. Also, the analysis considers two TTL implementations: TTL-IE and TTL-T. This is discussed in Chapter 3.

2- Proposal of a new replacement policy called FB-FIFO. This is explored in
Chapter 3\textsuperscript{9}.

3- Introduction of a new analytical model for estimating the IHR and IAHD of the HCS using LCE. LRU caching under TTL-IE or TTL-T is considered. Also, fixed as well as temporary leaf caches are considered. This is analyzed in Chapter 4.

4- Proposal of a new sharing protocol for the HCS called PCO. This is examined in Chapter 4\textsuperscript{10}.

5- Presentation of a new analytical model for estimating the IAHD for the DCS with fixed or temporary caches. LRU caching under TTL-IE or TTL-T is considered. This is discussed in Chapter 5\textsuperscript{11}.

1.4 Dissertation Roadmap

In this dissertation, the chapters are organized as follows. The background and related work are discussed in Chapter 2. The analysis of the proposed SCS is presented in Chapter 3. The analysis of the HCS and the DCS are introduced in Chapter 4 and Chapter 5, respectively. Chapter 6 concludes the dissertation and discusses future work. Finally, Appendix A and Appendix B include supplementary discussions for Chapter 3 and Chapter 4, respectively.

\textsuperscript{9}The work introduced in Chapter 3 under TTL-IE has been published in [12].

\textsuperscript{10}The work introduced in Chapter 4 for fixed leaf caches has been submitted to IEEE/ACM Trans. Netw. on Dec 2012.

\textsuperscript{11}The simulation work for DCS with TCs, assuming the batch-arrival-batch-departure connectivity model was published in [9, 10]. This work is not included in this dissertation for conciseness.
Chapter 2

Background

This chapter presents a brief background and discusses the related work for the considered cache systems. First, background for three main issues that affect the Web cache performance are discussed: (1) HyperText Transfer Protocol (HTTP) and cache consistency mechanisms are discussed in Section 2.1; (2) cache replacement policies are discussed in Section 2.2; and (3) the access pattern characteristics are discussed in Section 2.3. Then, Section 2.4 discusses the related analytical models in the literature for single cache systems. Sections 2.5 and 2.6 present the research on hierarchical and distributed cache systems, respectively. The research on mobile cache systems is discussed briefly in Section 2.7.

2.1 HTTP and Cache Consistency Mechanisms

The World Wide Web (Web) was invented by Tim Berners-Lee, a scientist at European Council for Nuclear Research (CERN), in 1989. The Web was originally developed to meet the demand for automatic information sharing between scientists working in different universities and institutes all over the world. The basic idea of the Web was to merge the technologies of personal computers, computer networking, and hypertext into a powerful and easy to use global information system [78]. The architecture of the Web uses the client-server model [79], where users access the Web objects over the Internet from origin servers (OSs) using Web browser software (e.g.
Hypertext Transfer Protocol (HTTP) is an application layer protocol that relies on the underlying TCP/IP protocols in order to provide reliable data transfer between Web users and the OSs\(^1\). Moreover, the HTTP requests and responses are encoded using headers to specify the interaction between the Web users, the OS, and the intermediate Web cache [80].

Fig. 2.1(a) shows a set of request and response headers for a new object that is not in cache (i.e. a cache miss). The location and the name of the requested object are specified using uniform resource locator (URL). GET method is used to retrieve a specific object indicated by the URL. Status Code 200 in the response indicates that the object is retrieved successfully from the OS. The response includes 119494 bytes of data for the object. According to the Expires header, the object will expire on (Mon, 25 Nov 2013 17:09:31 GMT). This means that the object life time (or the time-to-live (TTL) value on Mon, 25 Nov 2012 17:09:31 GMT) is one year. If this object is cached, the TTL value will decreases linearly with time until it reaches zero (i.e. object expires) on (Mon, 25 Nov 2013 17:09:31 GMT). The cache can serve this object without contacting the OS as long as the TTL value exceeds zero (i.e. as long as the object is fresh).

The semantics of the GET method change to “conditional GET” if the request message includes an If-Modified-Since header. For example, in Fig. 2.1(a), the Last-Modified header indicates that the last time when the object has been modified is

\(^1\)Web users may communicate with different types of servers (e.g. FTP). However, this dissertation is concerned with HTTP servers.
(a) Cache miss.

(b) Cache hit.

Figure 2.1: Sample HTTP request and response headers.

(Tue, 20 Nov 2012 19:52:07 GMT). Thus, the Last-Modified header can be used as a cache validator. If this object is requested after it expires (i.e. after Mon, 25 Nov 2013 17:09:31 GMT), then the cache uses the If-Modified-Since request header to ask the OS to send the object only if it was modified after (Tue, 20 Nov 2012 19:52:07 GMT). Otherwise, if the object was not modified (i.e. the cached object expired but it is still valid), then the OS does not send the object to the cache. In this case, the Status Code 304 in the response header indicates that the object is still valid and the cache can serve this object (i.e. a cache hit), as shown in Fig. 2.1(b).

The cache consistency problem (i.e. keeping the cached objects valid) has drawn attention as a key factor that affects the cache performance, especially for mobile
devices and wireless environments [36, 38, 81]. Cache consistency mechanisms are mainly classified into strong consistency mechanisms and weak consistency mechanisms. Strong consistency mechanisms, such as poll-each-read (PER) and call-back (CB) ensure that the cache always serves valid objects, but they generate considerable overhead traffic and increase the load on the OS [13, 38, 82, 83]. In PER, the cache attempts to retrieve the object from the OS at each access and the OS only replies to it with an acknowledgment when the cached object is valid. If the object is invalid, the server sends the updated object to the cache. In CB, the OS sends an invalidation message at each update to the caches that have invalid copies of the updated object. In this case, the OS has to track the locations of cached objects.

In weak consistency mechanisms, the cache validates the cached object periodically to reduce the overhead traffic with the OS and improve the user’s perceived latency. However, weak consistency mechanisms might cause the cache to serve invalid objects. Note that the cached copy of an object is valid if it is identical to the original copy at the OS, which cannot be guaranteed without contacting the OS [35, 37, 84, 85].

Weak consistency mechanisms are particularly useful for Web applications in which the user can tolerate (for a reasonable period of time) invalid objects. For example, it is useful (or at least not harmful) to retrieve cached copies of online newspapers, magazines, and personal home pages, even after their contents were updated at the OS. It has been shown that weak consistency is a viable and economical approach to deliver content that does not have a strict freshness requirement [37].

---

2In any browser software, a user always has the option to validate the Web objects by pressing reload/refresh button.
The concept of TTL [1] was introduced to support weak consistency as discussed in Section 1.1.1. In this dissertation, two implementations for the TTL mechanism are considered: TTL immediate ejection (TTL-IE) and typical TTL (TTL-T). The example in Fig. 2.1 shows the SCS under TTL-T, where the cache has to validate the expired object from the OS before serving it to the user. However, if the SCS adopts TTL-IE, the object will be ejected once it expires on (Mon, 25 Nov 2013 17:09:31 GMT). Fig. 2.2 shows an example where an object is accessed from a cache under TTL-T or TTL-IE. Note that, in TTL-T, if there is a high probability that the expired cached objects will be updated in the next request, it might be useful to eject the cached objects once they expire (i.e. use TTL-IE). This would not only avoid the communication overhead caused by the validation requests, but also would prevent
the cache from being polluted by invalid objects.

2.2 Cache Replacement Policies

The cache size is limited compared to the huge number of Web objects\(^3\), and it is only possible for a cache to store a fraction of the requested Web objects. Hence, cache performance depends greatly on the replacement policy that is used to select which objects to keep in cache in order to maximize the probability that future requests will be satisfied from the cache. The replacement policy decides which object is ejected from the cache to make room for a new object based on object characteristics, such as popularity, recency (i.e. how recent the object is requested), size, or expiry rate. The replacement policy may also account for network conditions, such as the bandwidth available between the cache and the OS \([12, 13, 14, 15, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103]\).

Some replacement policies consider only one object characteristic when making replacement decisions. For example, Perfect-LFU uses the popularity of objects (i.e. how many times the object has been requested) to decide which object to eject from the cache. Another example is Largest File First (LFF), also called SIZE, which bases the replacement decision on the size of the cached objects \([89]\).

Furthermore, replacement policies may consider more than one object characteristic. For example, LRFU \([92]\), LFU-Dynamic Aging (LFU-DA) \([94]\), and LRU-K \([98]\) consider both object popularity and recency. Another example is LRU-Threshold,

\(^3\)There are 3.2 billion Web objects as suggested by Wolman \textit{et al.} in 1999 \([11]\).
which considers both recency and size [89]. Also, replacement policies may consider both object characteristics and network conditions. For example, Greedy Dual (GD) [97] considers the object recency and the number of hops between the cache and the OS.

Several cache replacement policies were introduced to enhance the use of cache storage. The following list is a brief description of replacement policies in the literature:

- **Random (RAND):** ejects a random object from the cache.
- **First-In-First-Out (FIFO):** ejects the oldest cached object.
- **Least Frequently Used (LFU):** ejects the object with the fewest references. There are two versions of LFU:
  
  1. **Perfect-LFU:** maintains the number of references for each object, even if the object is not currently cached,
  
  2. **In-Cache-LFU:** maintains the number of references for cached objects only.

- **Least Recently Used (LRU):** ejects the least recently requested object.
- **Largest File First (LFF):** also called SIZE, ejects the largest object.
- **LFU-Aging [15]:** a variant of In-Cache-LFU, in which the request counts are divided by 2 when the average value of all counters exceed a certain threshold.
- **LFU* [96]:** a variant of In-Cache-LFU, in which only the cached objects with request count of one can be ejected from the cache. Hence, new requested objects do not always added to the cache.
• LRU-Threshold: a variant of LRU, in which no object larger than the Threshold is cached.

• LRU-k [98]: a variant of LRU, in which the cache maintain the last $k$ request times for each cached object. LRU-k ejects the object with the largest time elapsed since the $k$th last request was made. LRU is a special case of LRU-k, where $k=1$.

• GreedyDual-Size (GDS) [95]: ejects the object with the smallest value for a certain utility function that incorporates object size, recency, and the cost of retrieving the object from the OS.

• Adaptive Replacement Cache (ARC) [91]: In ARC, the cache maintains two LRU lists, with each has a length equals to the cache capacity$^4$. The first list is for one-time requested objects only. The second list is for objects that have been requested more than once. A fraction of the objects on these two lists are physically cached. The number of objects that are cached from the first list is determined by a threshold, $p$. ARC ejects the least recently requested cached object from the first list, if the number of cached objects that belong to the first list is greater than $p$. Otherwise, ARC ejects the least recently requested cached object from the second list.

2.3 Access Pattern Characteristics

The performance of the cache replacement policies depends greatly on the users’ access pattern (or the workload). For example, LRU outperforms Perfect-LFU when

$^4$The number of objects that can be cached simultaneously, assuming fixed object size.
the access pattern exhibits strong temporal locality\(^5\), due to correlation between requests \([6,93]\). However, this is not true under the independent reference model (IRM), where the object requests are independent and identically distributed.

Web access patterns have characteristics that were commonly used in developing tractable analytical models for estimating the performance of Web caches. These characteristics are summarized in the following list:

- The requests follow a Poisson process, where the inter-request times are independent and exponentially distributed \([96]\).
- Object popularity follows a Zipf-like distribution \([6,11,104]\).
- The number of objects at Web servers is growing continuously \([104]\).
- Object size distribution is heavy-tailed (Pareto) \([96,105]\)
- The expected lifetime (i.e. the initial TTL value) of an object is an exponential random variable, as suggested in \([11,74]\).
- No correlation between the object lifetime and its popularity \([79]\)
- Web traces exhibit strong temporal locality \([106]\). The relation between the object popularity and the temporal locality is discussed below in the remainder of this section.

There are two sources for the temporal locality: long-term and short-term popularity. The long-term popularity is determined by the Zipf-like distribution, such that the probability of the \(i\)th most popular object being selected from \(M\) objects is \(\frac{1}{\sigma i^\alpha}\), where \(\sigma = \sum_{i=1}^{M} \frac{1}{i^\alpha}\) and \(\alpha \in [0,1]\) is the Zipf slope. This is true regardless of when

\(^5\)The objects that are requested in the recent past are likely to be requested again in the near future.
object \( i \) was last requested. This is called independent reference model (IRM), where the identities of the requested objects are independent and identically distributed (i.i.d.) random variables \([107]\).

The short-term popularity (recency) occurs due to the temporal correlation between object requests. This is called a Markov reference model (MRM), where the probability of selecting an object \( i \) depends on when this object was requested in the near past \([108]\).

The study in \([6]\) showed that the temporal locality created due to the long-term popularity is similar to that observed in real traces. Thus, it is not necessary (at least for obtaining accurate qualitative results) to model the temporal correlations. A more careful study of real traces revealed that the long-term popularity alone does not suffice to fully capture the temporal locality exhibited by real traces \([93]\).

The studies in \([63,109]\) showed that the performance of a large cache does not depend on the correlation in the request traffic. Particularly, the studies in \([63,109]\) showed that the LRU cache performance is asymptotically identical to the case of IRM, if the cache is large. Furthermore, Jelenkovic et al. \([62]\) determined the smallest cache size beyond which this asymptotic insensitivity property holds.

### 2.4 SCS Analytical Models in the Literature

This section presents the related analytical models for the SCS proposed in Section 1.1.1. IRM with fixed-size objects was widely used to develop tractable analytical caching models, especially when a cache with finite size was considered. In this
section, all the presented analytical models use this assumption.

Wolman et al. [11] proposed an analytical model for estimating the steady state hit ratio (SSHR) for a single cache with infinite capacity (Infinite Cache). The analysis in [11] considered object lifetime, which is exponentially distributed and correlated with the object popularity. The study in [11] showed that the SSHR is very sensitive to the object expiry rate. The results in [11] also showed that the increase in the request rate, relative to the expiry rate, enhances the SSHR.

Breslau et al. [6] proposed an analytical model for estimating the SSHR of Infinite Cache and Perfect-LFU, assuming objects that do not expire. The results in [6] showed that the SSHR of Perfect-LFU increases logarithmically or as a small power as a function of cache size.

Analytical models for estimating SSHR of LRU and FIFO were also introduced in [8, 61, 62, 63, 64, 65, 66].

Gelenbe et al. [66] extended the analysis provided for FIFO in [99] in order to show that FIFO and RAND replacement policies reach the same SSHR. Dan et al. [65] developed approximate analytical models for estimating the SSHR of LRU and FIFO. The study in [65] showed that LRU always outperforms FIFO in steady state.

Jelenkovic et al. [64] showed that computing the LRU miss probability is the same as computing the move-to-front (MTF) search cost distribution. Also, the study in [61] proposed an analytical model for estimating the approximate SSHR of a single cache running LRU.

The analytical models introduced in [61, 62, 63, 64, 65, 66], however, assumed objects that do not expire. More realistic studies in [8, 38] provided an analytical model for
LRU cache under strong cache consistency model, where the cache has to ensure that the cached object is identical to the one at OS whenever the object is requested (i.e. poll-each-read (PER)). Also, Krishnamurthy et al. [110] studied the single LRU cache under TTL-T using trace-driven simulations.

While the analytical models in previous studies focused on estimating the SSHR, assuming stationary access pattern, to the best of my knowledge, the work in this dissertation is the first attempt to develop an analytical model to study the evolution of LRU and FIFO with time under TTL-based consistency.

2.5 Research on Hierarchical Cache Systems

Hierarchical Caching was first proposed in the Harvest project to share the interests of a large community of users, and has already been implemented in several countries [7, 111, 112, 113]. With hierarchical caching, caches are placed at different network levels. At the bottom level of the hierarchy are users’ caches. When a request is not satisfied by a user’s cache, the request is sent to the institutional cache. If the object is not in the institutional cache, the request travels to the regional cache, which in turn forwards unsatisfied requests to the national cache. If the object is not found at any cache level, the national cache directly contacts the OS. The main drawback of this scheme is that every hierarchy introduces additional delays. Moreover, higher level caches could get overwhelmed with requests from caches at lower levels, which causes higher level caches to introduce more delays [3, 7, 114, 115, 116, 117, 118].

As discussed in Section 1.1.2, cooperation between caches at different network
levels requires a sharing protocol. Section 2.5.1 presents the main sharing protocols proposed in the literature. Then, the HCS analytical models in the literature are discussed in Section 2.5.2.

2.5.1 HCS Sharing Protocols

Many sharing protocols were proposed in the literature to allow the cooperation between caches at different levels of the cache hierarchy. Leave copy everywhere (LCE) was developed in the Harvest project [112]. In LCE, when the object is found, either at a cache or at the OS, it travels down the hierarchy, and a copy is left at each of the intermediate caches. The main drawback of LCE is that it allows objects that are effectively one-timers\(^6\) to pollute the cache hierarchy. Moreover, LCE requires large caches at the high levels of the hierarchy (i.e. caches that are close to the OS) and these caches may become bottlenecks between the users and the OS [25, 26].

The studies in [24, 25, 26, 119] developed sharing protocols that achieve higher steady state average hit distance (SSAHD) than LCE under the assumption that the objects do not expire. Che et al. [25] introduced two design principles to enhance the LCE performance in a two-level LRU cache hierarchy. First, the root cache size has to be larger than the leaf cache size. Second, objects that are effectively one-timers should not be cached at all throughout the cache hierarchy.

Based on these principles, Che et al. [25] introduced a sharing protocol wherein a cache is allowed to store an object if the object is not effectively a one-timer with

\(^6\)A one-timer is an object that is requested once, and will never be requested again. The object is effectively a one-timer if its request rate is too low to generate a cache hit [25].
respect to this cache. Hence, the small leaf cache is preserved for the most popular objects (objects with high request rate, such that they generate cache hits for a small LRU cache), while the larger root cache may store objects with lower popularity. The objects that are effectively one-timers (with respect to the root cache) are not cached in any cache throughout the cache hierarchy. Also, to account for the situation when the object popularity decreases, if a cached object is ejected from the leaf cache, it will be sent to the root cache if the root cache does not have this object. The main problem in this protocol is that it requires the leaf cache to constantly track the access frequency of recently requested objects. Also, the average maximum object access interval without a cache miss has to be measured periodically for every cache in the cache hierarchy. This allows each cache to decide whether an object, with a certain request rate, is effectively one-timer or not.

Wong et al. [119] have proposed a sharing protocol for LRU cache hierarchy, called DEMOTE, where the leaf cache sends the ejected objects to the root cache. Additionally, when the root cache forwards an object to the leaf cache, it moves its local copy to the tail of its LRU list (to be ejected in the next replacement). The goal of DEMOTE is to avoid the duplication of the same objects at multiple levels.

Laoutaris et al. [24] introduced two sharing protocols: leave copy down (LCD) and move copy down (MCD). In LCD, when the object is found, it travels down the hierarchy and a copy is left only at the cache that is just below the level where the object is found. In MCD, when the object is found, it moves down the hierarchy to the cache that is just below the level where the object is found. Laoutaris et al. [24] showed by simulations that LCD outperforms LCE, DEMOTE, and MCD in terms of
SSAHD that the LRU cache hierarchy can achieve without the considerable overhead of the sharing protocol proposed by Che et al. [25].

In this dissertation, the design and analysis of a new sharing protocol, called PCO, is introduced. PCO achieves higher SSAHD than LCE and LCD. Also, like LCD, PCO is less complex than the sharing protocol proposed in [25], as will be discussed in detail in Chapter 4.

2.5.2 HCS Analytical Models in the Literature

Analytical models for estimating the steady state hit ratio (SSHR) for LCE under TTL-IE were introduced in [7, 37, 120]. A cache hierarchy based on a tree topology was analyzed in [7, 37], while Tang et al. [120] introduced an analytical model for unstructured P2P networks. These models assumed infinite sized caches. Thus, they generate an optimistic cache performance, since caches are limited in size compared to the huge number of Web objects [11].

The studies in [25, 41] introduced analytical models for an LRU cache hierarchy. The study by Che et al. [25] is the closest one to the proposed HCS analysis in Chapter 4. Che et al. used the mean field approximation approach for estimating the S SHR of a two-level cache hierarchy using LCE. Che et al. assumed that LRU runs locally at each cache and all the objects have the same size. These are the same assumptions adopted by the proposed analysis in Chapter 4. Moreover, Laoutaris et al. [41] introduced an analytical model for a two-level cache hierarchy using LCD based on the approximations proposed by Che et al. [25].

Compared to the proposed analysis in Chapter 4, the analytical models in [25]
and [41] have two drawbacks. First, they do not account for object expiry. Second, analytical models in [25] and [41] do not capture the transient behavior of the cache hierarchy under a non-stationary access pattern, as discussed in Section 1.3.

To the best of my knowledge, this dissertation introduces the first analytical model that characterizes the transient performance of an LRU cache hierarchy under TTL-IE or TTL-T. Moreover, this dissertation proposes the first analytical model that considers an LRU cache hierarchy with temporary leaf caches (TLCs), as discussed in Section 1.1.4.

2.6 Research on Distributed Caching Systems

In distributed caching, no intermediate caches are set up, and there is only one level of caches (e.g. users’ caches, or institutional caches) that cooperate to serve each others’ misses. Distributed caching offers some different advantages than hierarchical caching, including a lower access latency, a higher degree of fault tolerance, and a more uniform load distribution [3, 7, 121, 122, 123, 124, 125, 126, 127].

Plenty of sharing protocols were proposed in the literature to allow the cooperation between sibling caches in a DCS, as discussed in Section 1.1.3. In Section 2.6.1, the main DCS sharing protocols are described. Then, Section 2.6.2 presents the DCS analytical models in the literature.

2.6.1 DCS Sharing Protocols

Internet caching protocol (ICP) was designed for the NLANR project [7, 44, 111]. In ICP, the sibling cache cannot resolve another cache request, if the sibling cache does
not have the requested object. Only cache hits are resolved by the nearest sibling cache [29, 128]. In ICP, the cache distinguishes between the local requests and the remote requests, and the caches in the DCS do not coordinate their replacement policies, as discussed in Section 1.1.3. Note that ICP does not improve the performance of individual caches. On the other hand, the users experience better performance due to remote cache hits. This requires multicasting the queries, and result in ICP generating considerable traffic on every cache miss. Also, ICP results in long connection time to find a remote cache with the requested object [7].

Cache Summary, developed by Fan et al. [44], and Cache Digest, developed by Rousskov et al. [129], avoid the drawbacks of ICP by allowing caches in a DCS to periodically exchange information about their contents. Moreover, to achieve more efficient and scalable DCS, these protocols may utilize a hierarchical infrastructure for storing meta-data about the locations of objects in the DCS [7]. Note that the hierarchical infrastructure is used only for distributing the locations of cached objects in the DCS, rather than caching the objects themselves. Also, note that ICP, Cache Summary, and Cache Digest protocols allow multiple copies of the same objects to be stored at different locations within the DCS.

Cache array routing protocol (CARP) was introduced by Valloppillil et al. [45], where at most one copy of an object can be stored in the DCS at a time\(^7\). In CARP, the URL-space is divided among the caches in the DCS. Thus, the request for the same URL is always forwarded to the same cache. In CARP, a local cache does not store the object that is served from the remote cache in the DCS. Also, like ICP,

\(^7\)This is also called hash-based caching, single copy caching, or one-copy heuristic (1CH).
the replacement policy is also applied on individual caches independently. However, when an object is requested, the remote cache may increase the caching priority for this object [44,130,131].

Global Cache [44] is another approach that prevents the redundant caching of the same object. Global Cache is a hypothetical approach, where it is assumed that the caches in the DCS coordinate their replacement policy and appear as one unified cache, with a cache size that is equal to the aggregate cache sizes in the DCS. In this case, if the local cache is full and it needs to store more objects, it may use the available cache capacity on another cache in the DCS. If all caches in the DCS are full, then the replacement policy is applied across all caches in DCS to decide which objects to eject. In this case, at most one copy of an object may exist in the DSC. Hence, the analysis for the SCS introduced in Chapter 3 can be used for estimating the IHR of the DCS using Global Cache.

In Chapter 5, the proposed DCS using ICP under TTL-based consistency is evaluated. I believe that the proposed analysis of ICP can be extended to CARP. However, this will be considered in future work.

2.6.2 DCS Analytical Models in the Literature

Analytical models for estimating the steady state hit ratio (SSHR) of the DCS under TTL-IE were introduced in [7]. In [7], it is assumed that the caches exchange their meta-data about their contents instantaneously. It is also assumed that the caches have infinite size.

Moreover, the study in [40] introduced an analytical model for estimating the
SSHR of the DCS in the context of LRU. The model proposed in [40] is based on the analysis proposed by Dan et al. [65]. The study in [40] also considered a DCS with dependent-LRU, where the local LRU list is updated because of remote hits. However, the study in [40] assumed objects that do not expire.

To the best of my knowledge, the work in this dissertation is the first attempt to introduce analysis for estimating the IHR and the IAHD of the DCS running LRU under TTL-based consistency. This will be discussed in Chapter 5. Also, possible extension of the proposed analysis for a DCS with dependent-LRU will be discussed briefly in Chapter 6.

2.7 Research on Mobile Cache Systems

In Mobile Ad-Hoc Networks (MANETs), cooperative caching on mobile devices is essential to reduce the usage of limited bandwidth, save battery life, and improve the user’s perceived latency [9, 10, 13, 48, 49, 122, 132, 133]. Furthermore, caching on vehicles is promising since they can provide more processing power and enough space to accommodate a cache with large storage size, unlike traditional mobile devices (e.g. smart phones) [36, 50, 134]. On the other hand, cooperative caching schemes must account for disconnections of mobile devices, due to mobility, as well as the limited battery life of mobile devices [9].

Cooperative caching in MANET was considered in many studies [35, 71, 135, 136, 137, 138, 139, 140, 141, 142], as will be discussed briefly in the remainder of this section. However, to the best of my knowledge, the work in this dissertation is the first attempt
for estimating the performance of cooperative caching on mobile devices that run LRU under TTL-based consistency.

The study in [135] introduced three cooperative caching systems: CacheData, CachePath, and HybridCache. In CacheData, the node (mobile user) caches the passing-by popular objects locally in order to serve future requests. In CachePath, intermediate nodes between the requesting node and the origin server (OS) store the location (path information) where the object is cached (i.e. the location of the requesting node). Later, when this object is requested from one of those intermediate nodes, the request will be redirected to that location, rather than the OS, which reduces the response delay. Note that multiple copies of the same object or object path might be cached in the MANET, since each node makes the cache decision independently. The simulation results in [135] showed that CachePath outperforms CacheData when the cache size is small or the object expiry rate is low, while CacheData performs better in other situations. Hence, HybridCache was introduced in [135] to take advantage of both CacheData and CachePath. Particularly, when a node forwards an object, it may cache the object or path based on some criteria, such as object size and object expiry rate.

The main disadvantage of the systems introduced in [135] is that the cached object (or path information) will only be accessed if it is in the path of the request to the OS. Hence, the cache hit ratio depends greatly on the node positions relative to each other and relative to the access point. The system introduced by Lim et al. [136] overcomes this problem. In [136], the node broadcasts the request to all other nodes in the network, then the requesting node retrieves the object from the first acknowledging
node (which has the required object) it hears from. Although this approach suffers from high bandwidth usage due to the broadcasts, it is simple and achieves a low latency.

Artail et al. [71], proposed a cooperative and adaptive caching system (COACS) for MANETs. COACS consists of three types of nodes (mobile users): caching nodes that cache previously requested objects, query directories that index the cached objects by holding the queries along with the locations of the corresponding caching nodes (i.e. store the path information like CachePath), and requesting nodes (any node, including a query directory or a caching node). In COACS [71], when a node requests an object that is not cached by any caching node in the system, the OS is accessed to retrieve this object. Upon receiving the object, the node that requested the object will act as a caching node by caching this object. The nearest query directory to the caching node will cache the query and make an entry in its hash table to link the query to its response.

Like the approach in [136], all the caching nodes in COACS have to be checked for the requested object before sending the request to the OS. However, instead of broadcasting the request, the request in COACS is forwarded from a query directory to another until the object is found or until all the query directories are checked. Hence, COACS reduces the bandwidth usage due to the broadcasts used in [136].

The simulation results in [71] showed that COACS outperforms both CacheData and CachePath. Furthermore, the study in [71] provides analysis for estimating the upper and lower limits of average response delay, by setting the average hit ratio to one and zero, respectively. However, they did not provide any expression for calculating
the average hit ratio. Also, the proposed work in [71] did not consider the cache consistency problem. Later in [35], a cache consistency mechanism that is built on top of COACS has been proposed.

In the proposed cooperative caching system in this dissertation, two types of nodes (mobile users) are assumed: caching nodes (mobile device hosting caches) and requesting nodes (any node in the MANET including the caching nodes). Like COACS [71], all the caching nodes in the MANET have to be checked for the requested object before sending the request to the OS. However, the proposed system does not need query directories. Note that, in the proposed HCS, the MANET contains only one caching node (i.e. the temporary leaf cache (TLC)), while in the proposed DCS, the requesting node may only query the nearest caching node in the MANET (i.e. the local TC). Also, similar to the approach proposed in [136], the local TC broadcasts the request to the other caching nodes in the MANET (i.e. remote TCs), if it does not have the requested objects.

Note that the work in this dissertation is not concerned with evaluating the MANET routing protocols that are used to determine the multihop communication path, between the mobile user and the location of the requested object, that minimizes the energy consumption and the response delay [143,144,145]. In the proposed HCS the mobile user communicates only with its TLC, which has a direct link to the access point. Also, in the DCS, it is assumed that any mobile user communicates only with its local TC, which has a direct link to the access point, or to any remote TC within the MANET, as will be discussed in detail in Section 5.2.
Chapter 3

Analysis of the Single Cache System (SCS)

This chapter introduces a novel analytical model for characterizing the hit ratio of a single cache system (SCS) as a function of time, as discussed in Section 1.2. The considered single cache can be thought of as a Web cache that is installed at the ISP, or at the main router of an institution (e.g. university, company, etc.) as shown in Fig 1.1. Also, it might be a local cache installed at a Web user (e.g. browser cache). The proposed analysis considers two implementations for TTL consistency: typical TTL (TTL-T) and TTL immediate ejection (TTL-IE). As discussed in Section 1.1.1, the analysis also considers the case where objects do not expire, which is called non-expiry model (NEM).

The proposed analysis models the activity of a single cache with infinite capacity (Infinite Cache) or finite capacity. The cache capacity is defined as the number of objects that can be stored in a cache simultaneously, assuming that all objects have the same size. For finite cache capacity, the proposed analysis considers two replacement policies: FIFO and LRU. Also, steady state analysis of Perfect-LFU is discussed. Note that the proposed analysis in this chapter can be extended for cooperative cache systems (e.g. hierarchical caching), as will be discussed in the subsequent chapters in this dissertation.

Moreover, this chapter introduces a new replacement policy, called frequency-based-FIFO (FB-FIFO). FB-FIFO improves the instantaneous hit ratio (IHR) by
creating a variable-sized protected cache segment for objects that are not effectively one-timers, as will be discussed in detail in this chapter. Note that no analysis is provided for FB-FIFO.

The remainder of this chapter is organized as follows. First, the analysis assumptions are discussed in Section 3.1. Second, assuming a stationary access pattern, the proposed analysis for Infinite Cache, Perfect-LFU, FIFO, and LRU is presented in Sections 3.2, 3.3, 3.4, and 3.5, respectively. Third, the proposed analysis is generalized for a non-stationary access pattern in Section 3.6. Next, FB-FIFO is presented in Section 3.7. Section 3.8 is an evaluation for the IHR of a single cache under NEM and TTL-IE, while Section 3.9 is a steady state evaluation for a single cache under TTL-IE and TTL-T. Finally, Section 3.10 summarizes the findings of this section.

3.1 Analysis Assumptions

The proposed analysis assumes that the origin server (OS) has \( M(t) \) objects at time \( t \). Allowing the number of OS objects to change with time allows the analysis to accommodate the continuous generation of new popular objects [12, 74, 75].

Consider the interval \((0, t_v)\) or \((t_v, \infty)\) in Fig. 1.10(b) where the number of objects is fixed, such that \( M(t) \approx M \). Users request those \( M \) objects according to a Poisson process with rate \( \beta [146] \). The probability of each request being for one of the \( M \) objects is determined by a Zipf-like distribution [6, 11]. In this distribution, the probability of the \( i \)th most popular object being selected is \( 1/(\sigma i^\alpha) \), where \( \sigma = \sum_{i=1}^{M} 1/i^\alpha \) and \( \alpha \) is the Zipf slope such that \( 0 < \alpha \leq 1 \). Note that the probability of
requesting an object \( i \) is independent of past history (i.e. the independent reference model (IRM) is assumed) [93, 99, 108].

The overall Poisson request process can be modeled as a sum of \( M \) independent Poisson processes, with each representing the requests for one of the \( M \) objects [146]. The average request rate of the Poisson process for object \( i \in [1, M] \) at time \( t \), is

\[
\lambda(i) = \frac{\beta}{\sigma_i^{\alpha}}
\]

The proposed analysis assumes that the initial TTL value of a cached object, \( i \), is exponentially distributed with mean \( 1/\mu(i) \) that is independent from object popularity, as suggested in [11, 13, 35, 74].

It is assumed that \( \alpha, \beta, \) and \( \mu(i) \) are not a function of time. However, the proposed analysis can be extended to cover these cases, which are excluded from this chapter merely for conciseness\(^1\).

In this chapter, the cache capacity, \( C \), is defined as the number of objects that can be cached simultaneously (i.e. \( C \) is the cache size divided by the object size). For simplicity, it is assumed that the cache can store up to \( C \) objects, where the objects are of the same size.

Note that assuming the IRM with fixed-size objects is widely used to develop tractable analytical cache models, especially when the cache has finite size [13, 24, 25, 35, 38]. However, these assumptions might not accurately model empirical traces. Thus, an interesting extension of the proposed model would be to incorporate tem-

\(^1\)In Chapter 4, the case where the request rate at the cache is a function of time will be considered.
poral correlations, as well as assuming objects with different sizes.

In the following sections, continuous time Markov chain analysis [147] is used to calculate the IHR for $M$ objects, $H(t)$. Since the Poisson request processes for each of the $M$ objects are independent, it is possible to start by analyzing the probability that the request for object $i$ at time $t$ is not served by the cache, $S(t, i)$. Then, $H(t)$ will be calculated as

$$H(t) = \sum_{i=1}^{M} \frac{1 - S(t, i)}{\sigma^i \alpha}$$  (3.1)

Assuming a stationary access pattern, analytical solutions for Infinite Cache, Perfect-LFU, FIFO, and LRU are developed in Section 3.2, Section 3.3, Section 3.4, and Section 3.5, respectively. Then, the analysis is extended for a non-stationary access pattern in Section 3.6.

3.2 Analysis of Infinite Cache

This section presents the analysis of a single Infinite Cache assuming a stationary access pattern, as shown in Fig. 1.10(a). First, the analysis for non-expiry model (NEM) is discussed in Section 3.2.1. Second, the analysis for TTL immediate ejection (TTL-IE) is discussed in Section 3.2.2. Third, the analysis for typical TTL (TTL-T) is discussed in Section 3.2.3.

Note that estimating the steady state hit ratio (SSHR) of Infinite Cache has been done under TTL-IE in [11], and the contribution of this section is providing an analytical model for estimating the IHR, as well as considering TTL-T.
3.2.1 Infinite Cache under NEM

In NEM, the objects do not expire, and thus, cached objects are always fresh. Hence, the Infinite Cache does not eject an object once it is cached. The state of the cache for an object \( i \) can be modeled using a Markov chain, as shown in Fig. 3.1. In Fig. 3.1, there are two states:

- Non-cache state (NCS): state 0, which corresponds to when object \( i \) is not in the cache.

- Cache-fresh state (CFS): state 1, which corresponds to when object \( i \) is in the cache.

In Fig. 3.1, the rate at which the object moves from state 0 to state 1 is equal to the object request rate (i.e. \( \lambda(i) \)), and the process does not return to state 0. Thus, the Markov chain flow matrix of object \( i \), \( Q(i) \), is given by

\[
Q(i) = \begin{bmatrix}
-\lambda(i) & \lambda(i) \\
0 & 0
\end{bmatrix}
\]

Assuming an empty cache at time \( t = 0 \), the probability matrix of object \( i \) at time
\( P(t, i) \), is calculated using (3.2) [146, 147].

\[
P(t, i) = \exp(Q(i) t)
\]  

(3.2)

The probability that a request for object \( i \) at time \( t \) will not be served from the cache, \( S(t, i) \), is equal to the probability that object \( i \) is in state 0 at time \( t \), \( P(t, i, 0) \). Hence, \( S(t, i) \), is calculated using (3.3) [146].

\[
S(t, i) = P(t, i, 0) = [P(t, i)]_{1,1} = e^{-\lambda(i)t}
\]  

(3.3)

where the notation \([.]_{r,c}\) denotes the element in row \( r \) and column \( c \) of a matrix [146].

The IHR is then calculated using (3.1). Note that, assuming a stationary access pattern, \( \lim_{t \to \infty} S(t, i) = 0 \), and the SSHR is

\[
\lim_{t \to \infty} H(t) = \sum_{i=1}^{M} \frac{\lambda(i)}{\beta} = 1
\]  

(3.4)

3.2.2 Infinite Cache under TTL-IE

In TTL-IE, objects are merely ejected when they expire. The state of the cache with respect to an object \( i \) can be modeled using a Markov chain, as shown in Fig. 3.2.

Like Fig. 3.1, the Markov chain in Fig. 3.2 has two states:

- State 0 (NCS): corresponds to when object \( i \) is not in the cache.
- State 1 (CFS): corresponds to when object \( i \) is in the cache.

In Fig. 3.2, the rate at which the object moves from state 0 to state 1 is equal to the object request rate (i.e. \( \lambda(i) \)) and the rate at which the process returns to state
0 is equal to the object expiry rate (i.e. $\mu(i)$). Hence, the Markov chain flow matrix of object $i$, $Q(i)$, is given by

$$Q(i) = \begin{bmatrix} -\lambda(i) & \lambda(i) \\ \mu(i) & -\mu(i) \end{bmatrix}$$

Assuming an empty cache at time $t = 0$, the probability matrix of object $i$ at time $t$, $P(t, i)$, is calculated using (3.2). Then, $S(t, i)$ is calculated as

$$S(t, i) = P(t, i, 0) = [P(t, i)]_{1,1}$$

(3.5)

where $[P(t, i)]_{1,1}$ is calculated using (3.6) [146].

$$[P(t, i)]_{1,1} = \frac{\mu(i)}{\lambda(i) + \mu(i)} + \frac{\lambda(i)}{\lambda(i) + \mu(i)}e^{-(\lambda(i)+\mu(i))t}$$

(3.6)

The IHR is then calculated using (3.1). Also, $\lim_{t \to \infty} S(t, i)$ can be calculated as

$$\lim_{t \to \infty} S(t, i) = \frac{\mu(i)}{\lambda(i) + \mu(i)}$$

(3.7)

Hence, the SSHR is
3.2.3 Infinite Cache under TTL-T

In TTL-T, unlike TTL-IE, the object is not ejected from the cache when it expires. The cache just marks this object as “expired”. Compared to TTL-IE, the Markov chain under TTL-T has an extra state that is added to accommodate object $i$ when it expires, as shown in Fig. 3.3. The Markov chain in Fig. 3.3 has three states:

- Non-cache state (NCS): state 0, corresponds to when object $i$ is not in the cache.
- Cache-fresh state (CFS): state 1, corresponds to when object $i$ is in the cache and is fresh.
- Cache-expired state (CES): state 2, corresponds to when object $i$ is in the cache and is expired.

In Fig. 3.3, the rate at which the object moves from state 0 to state 1 is equal to the object request rate (i.e. $\lambda(i)$), and the process does not return to state 0. When an object expires, it moves from state 1 to state 2 with a rate that is equal to the object expiry rate (i.e. $\mu(i)$). The object returns from the current CES to the CFS,

$$
\lim_{t \to \infty} H(t) = \sum_{i=1}^{M} \frac{\lambda(i)}{\beta} \left( \frac{\lambda(i)}{\lambda(i) + \mu(i)} \right)
$$

(3.8)
if it is updated with a fresh copy from the OS. This occurs when the expired object in the CES is invalid (i.e. the cached copy is not identical to the original copy on the OS), such that the request for this object (i.e. the validation request) is served from the OS, and thus, is counted as a cache miss, as discussed in Section 1.1.1. Therefore, object $i$ moves from the current CES to the CFS with rate

$$\varsigma(i) = \eta \lambda(i)$$

where $\eta$ is the validation factor (i.e. the probability that the validation request is served from the OS). Hence, the Markov chain flow matrix of object $i$, $Q(i)$, is given by

$$Q(i) = \begin{bmatrix}
-\lambda(i) & \lambda(i) & 0 \\
0 & -\mu(i) & \mu(i) \\
0 & \varsigma(i) & -\varsigma(i)
\end{bmatrix}$$

Assuming an empty cache at time $t = 0$, $P(t, i)$ is calculated using (3.2). Since the object is not served from the cache either when it is in the NCS, or when the object is in the CES and is invalid (with probability $\eta$), $S(t, i)$ is calculated as follows

$$S(t, i) = P(t, i, 0) + \eta P(t, i, 2)$$

$$= [P(t, i)]_{1,1} + \eta [P(t, i)]_{1,3} \quad (3.9)$$

where $P(t, i, j)$ is the probability that object $i$ is in state $j$ at time $t$. The IHR is then
calculated using (3.1). Note that, under TTL-T, \( \lim_{t \to \infty} S(t, i) \) can be calculated as

\[
\lim_{t \to \infty} S(t, i) = \frac{\eta \mu(i)}{\varsigma(i) + \mu(i)} \quad (3.10)
\]

where \( \lim_{t \to \infty}[P(t, i)]_{1,1} = 0 \) and \( \lim_{t \to \infty}[P(t, i)]_{1,3} = \mu(i)/\varsigma(i) + \mu(i) \). Hence, the SSHR of Infinite Cache under TTL-T is calculated as

\[
\lim_{t \to \infty} H(t) = \sum_{i=1}^{M} \frac{\lambda(i)}{\beta} \left( 1 - \frac{\eta \mu(i)}{\varsigma(i) + \mu(i)} \right) \quad (3.11)
\]

Also, note that the average validation request rate can be calculated as

\[
\nu(t) = \sum_{i=1}^{M} \lambda(i) P(t, i, 2)
\]

### 3.3 Analysis of Perfect-LFU

This section discusses how the analysis of Infinite Cache, introduced in Section 3.2, can be modified to estimate the SSHR of a single Perfect-LFU cache. No analysis is provided for estimating the IHR for Perfect-LFU.

The SSHR is estimated for a single Perfect-LFU cache under NEM, TTL-IE, and TTL-T by replacing the number of objects, \( M \), with the cache capacity, \( C \), in (3.4), (3.8), and (3.11), respectively\(^2\). Hence, the SSHR of a single Perfect-LFU cache under NEM, TTL-IE, and TTL-T is calculated using (3.12), (3.13), and (3.14),

\(^2\)This idea is used in [6] for estimating the SSHR of Perfect-LFU under NEM.
respectively. Note that the objects in $[1, C]$ are the most popular objects.

\[
\lim_{t \to \infty} H(t) = \sum_{i=1}^{C} \frac{\lambda(i)}{\beta} \tag{3.12}
\]

\[
\lim_{t \to \infty} H(t) = \sum_{i=1}^{C} \frac{\lambda(i)}{\beta} \left( \frac{\lambda(i)}{\lambda(i) + \mu(i)} \right) \tag{3.13}
\]

\[
\lim_{t \to \infty} H(t) = \sum_{i=1}^{C} \frac{\lambda(i)}{\beta} \left( 1 - \frac{\eta \mu(i)}{\varsigma(i) + \mu(i)} \right) \tag{3.14}
\]

Note that, like Infinite Cache, \( \lim_{t \to \infty} S(t, i) \) for Perfect-LFU under TTL-IE and TTL-T is calculated using (3.7) and (3.10), respectively.

Also, for Perfect-LFU under TTL-T, the average validation request rate is calculated as follows

\[
\lim_{t \to \infty} \nu(t) = \sum_{i=1}^{C} \lambda(i) \lim_{t \to \infty} P(t, i, 2) = \sum_{i=1}^{C} \lambda(i) \frac{\mu(i)}{\varsigma(i) + \mu(i)}
\]

3.4 Analysis of FIFO

In this section, a new analytical model for a single FIFO cache is presented. The proposed analysis assumes a stationary access pattern, as shown in Fig. 1.10(a). First, the analysis for NEM is discussed in Section 3.4.1. Then, the analysis for TTL-IE and TTL-T is discussed in Sections 3.4.2 and 3.4.3, respectively.
3.4.1 FIFO Cache under NEM

In FIFO, when a new object is requested, the oldest cached object is ejected, if the cache is full. The state of the cache with respect to an object $i$ can be modeled using the Markov chain shown in Fig. 3.4. Similar to the analysis in Section 3.2.1, the Markov chain in Fig. 3.4 has two types of states:

- Non-cache state (NCS): state 0, corresponds to when object $i$ is not in the cache.
- Cache-fresh states (CFSs): states 1 to $C$, correspond to when object $i$ is in the cache.

The CFSs represent the relative time at which object $i$ was brought into the cache. State $C$ corresponds to the most recently cached object. The recency of a cached object decreases as the object goes from CFS $j$ to the CFS $j - 1$. The CFS $j = 1$ corresponds to the oldest object in the cache.

In Fig. 3.4, an object $i$ moves from the NCS to state $C$ with rate $\lambda(i)$ (i.e. when the object is requested, but not currently cached). Furthermore, an object $i$ moves from the current state to the next lower-numbered one whenever a new object is introduced into the FIFO cache. Thus, the rate at which a cached object $i$ moves from the current CFS to the previous state, $\epsilon(t, i)$, can be calculated as
\[ \epsilon(t, i) = \sum_{m=1, m \neq i}^{M} \lambda(m) P(t, m, 0) \] (3.15)

where \( P(t, i, j) \) is the probability that object \( i \) is in state \( j \) at time \( t \). The flow matrix for the Markov chain in Fig. 3.4 is

\[
\begin{bmatrix}
-\lambda(i) & 0 & \ldots & 0 & 0 & \lambda(i) \\
\epsilon(t, i) & -\epsilon(t, i) & \ldots & 0 & 0 & 0 \\
: & : & : & : & : & : \\
0 & 0 & \ldots & \epsilon(t, i) & -\epsilon(t, i) & 0 \\
0 & 0 & \ldots & 0 & \epsilon(t, i) & -\epsilon(t, i)
\end{bmatrix}
\] (3.16)

Assuming an empty cache at time \( t = 0 \), \( P(t, i) \) is then calculated using (3.2), and \( S(t, i) \) is calculated using (3.5) (i.e. \( S(t, i) = P(t, i, 0) \)). Then, the IHR, \( H(t) \), is calculated using (3.1).

However, from (3.15) and (3.5), \( H(t) \) cannot be calculated without having the probabilities \( P(t, i, 0), \forall i \in [1, M] \), which are initially unknown. To overcome this problem, the following analysis is proposed based on the contraction mapping theory [32,33,34].

The proposed analysis starts by initializing \( P^0(t, i, 0), \forall i \in [1, M] \) to 1. Thus, \( P^0(t, i, j)|_{j>0} \) is initialized to 0, such that \( \sum_{j=0}^{C} P^0(t, i, j) = 1 \). Thus, (3.15) is rewritten for the first iteration of the proposed analysis, \( z = 1 \), as
Table 3.1: Iterative analysis for estimating the IHR of a single FIFO cache under NEM.

1- Set: $P^0(t, i, 0) = 1, \forall i \in [1, M]$
2- Set: $z = 0$
3- Do
4- $z = z + 1$
5- Calculate: $\epsilon^z(t, i), \forall i \in [1, M]$ using (3.18)
6- Calculate: $P^z(t, i), \forall i \in [1, M]$ using (3.19)
7- Calculate: $S^z(t, i), \forall i \in [1, M]$ using (3.20)
8- Calculate: $H^z(t)$ using (3.21)
9- While ($H^z(t) \neq H^{z-1}(t)$)
10- Output $H(t) = H^*(t) = H^z(t)$

\[
\epsilon^1(t, i) = \sum_{m=1, m\neq i}^{M} \lambda(m)
\]

(3.17)

where the superscript denotes the iteration at which the variable is calculated. Then, the flow matrix at $z = 1$, $Q^1(t, i)$, is established using $\epsilon^1(t, i)$ according to (3.16). Then, the probability matrix $P^1(t, i)$ is calculated using (3.2). Afterwards, $S^1(t, i)$ is calculated using (3.5), and $H^1(t)$ is calculated using (3.1). This procedure is repeated, and the analysis approaches the exact solution for the IHR iteratively and terminates after reaching the fixed point $H^*(t)$, as shown in Table 3.1. Note that the superscript $(.)^*$ indicates the final fixed point state probability solution provided by the contraction mapping analysis. Appendix A provides a brief background for the contraction mapping theory and discusses the convergence of the proposed analysis for FIFO and LRU.

In Table 3.1, at iteration $z$ of the contraction mapping analysis, $H^z(t)$ is calculated as follows. First, $\epsilon^z(t, i)$ is calculated as
\[ \epsilon^z(t, i) = \sum_{m=1, m \neq i}^{M} \lambda(m) P^{z-1}(t, m, 0) \]  

(3.18)

where \( P^0(t, m, 0) = 1 \) \( \forall i \in [1, M] \), and the flow matrix is calculated as

\[
Q^z(t, i) = \begin{bmatrix}
-\lambda(i) & 0 & \ldots & 0 & 0 & \ldots & \lambda_i \\
\epsilon^z(t, i) & -\epsilon^z(t, i) & \ldots & 0 & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & \epsilon^z(t, i) & -\epsilon^z(t, i) & 0 & \ldots \\
0 & 0 & \ldots & 0 & \epsilon^z(t, i) & -\epsilon^z(t, i) & \ldots \\
\end{bmatrix}
\]

Then, assuming an empty cache at time \( t = 0 \), \( P^z(t, i) \) is calculated as

\[ P^z(t, i) = \exp(Q^z(i) \times t) \]  

(3.19)

and \( S(t, i)^{z+1} \) is calculated as

\[ S^{z+1}(t, i) = [P^z(t, i)]_{1,1} \]  

(3.20)

Finally, the proposed analysis generates \( H^z(t) \) by calculating

\[ H^z(t) = \sum_{i=1}^{M} \frac{\lambda(i)}{\beta} (1 - S^z(t, i)) \]  

(3.21)

### 3.4.2 FIFO Cache under TTL-IE

For estimating the IHR of a single FIFO cache under TTL-IE, the same analysis introduced in Section 3.4.1 is used. However, the proposed analysis for TTL-IE
incorporates the object expiry factor into the proposed Markov chain in Fig. 3.4.

Hence, two changes are made to the NEM calculations. First, since under TTL-IE the object is ejected once it expires, the rate at which an object $i$ moves from the current CFS $j > 1$ to the NCS is equal to $\mu(i)$, as shown in Fig. 3.5. An object $i$ moves from CFS $j = 1$ to the NCS with a rate equal to $\mu(i) + \epsilon_z(t, i)$.

Second, an object $i$ moves from the current CFS $j < C$ to the next CFS $j + 1 \leq C$, if another object $m \neq i$ that currently occupies a CFS $c > j$, expires. Thus, an object $i$ moves from the current CFS to the next CFS with a rate equal to $\gamma_z(t, i, j)$, where $j \in [1, C - 1]$, such that

$$\gamma_z(t, i, j) = \sum_{m=1, m \neq i}^{M} \mu(m) \sum_{c=j+1}^{C} P^{z-1}(t, m, c) \quad (3.22)$$

where $p^z(t, i, j)$ is the probability that object $i$ is in state $j$ at time $t$. The proposed analysis in Table 3.1 can be used to calculate the IHR of a FIFO cache under TTL-IE. However, note that $\gamma_z(t, i, j)$ is also calculated along with $\epsilon_z(t, i)$ in step 5.
3.4.3 FIFO Cache under TTL-T

For FIFO under TTL-T, expired objects are not ejected from the cache once they expire. Rather, when an object $i$ expires, the cache just marks it as an expired object. Hence, the state of object $i$ is modeled using the Markov chain in Fig. 3.6, where each cache-fresh state (CFS) is associated with a cache-expired state (CES). In Fig. 3.6, there are three types of states:

- Non-cache state (NCS): state 0, corresponds to when object $i$ is not in the cache.
- Cache-fresh states (CFSs): odd states 1, 3, ..., $2C - 1$, which correspond to when object $i$ is in the cache and the object is fresh.
- Cache-expired states (CESs): even states 2, 4, ..., $2C$, which correspond to when object $i$ is in the cache and the object is expired.

In Fig. 3.6, the total number of cache states is $2C$. The state $2C - 1$ is occupied by the least recent object that entered the cache and is still fresh. Thus, when an object $i$ is in state 0, it moves to state $2C - 1$ with rate $\lambda(i)$. An object $i$ moves from the current CFS $j$ to the associated CES $j + 1$, when it expires. The expired object returns from CES $j + 1$ to CFS $j$ when it is updated from the OS. Thus, an object $i$ moves from a CFS to the associated CES with rate $\mu(i)$, and returns from the CES to the associated CFS with rate $\varsigma(i) = \eta \lambda(i)$. Note that the validation factor, $\eta$, is the probability that the request for an expired cached object (i.e. the validation request) will be served from the OS (i.e. the object will be updated and the request will be counted as a cache miss, as discussed in Section 1.1.1).

Furthermore, an object $i$ moves from the current CFS to the previous CFS (e.g. from
state 3 to 1), or moves to the NCS (i.e. from state 1 to 0), when a cache miss occurs. Thus, object \(i\) moves to the previous CFS, or to the NCS, with rate \(\epsilon^z(t, i)\). Similarly, object \(i\) moves from the current CES to the previous CES, or the NCS, with a rate equal to \(\epsilon^z(t, i)\).

For estimating the IHR of a single FIFO cache under TTL-T, the analysis introduced in Table 3.1 is used, where \(\epsilon^z(t, i)\), \(P^z(t, i)\), and \(H^z(t)\) are calculated using (3.18), (3.19), and (3.21), respectively. On the other hand, \(S^z(t, i)\) is calculated after modifying (3.20), such that

\[
S^z(t, i) = P^z(t, i, 0) + \eta \sum_{j=1}^{C} P^z(t, i, 2j) \tag{3.23}
\]

where \(P^z(t, i, j)\) is calculated as

\[
P^z(t, i, j) = [P^z(t, i)]_{1, j+1} = [\exp(Q^z(t, i, t))]_{1, j+1} \tag{3.24}
\]
Also, the average validation request rate can be calculated at iteration $z$ as

$$\nu^z(t) = \sum_{i=1}^{M} \lambda(i) \sum_{j=1}^{C} P^z(t, i, 2j)$$  \hspace{1cm} (3.25)

3.5 Analysis of LRU

In this section, a new analytical model of a single LRU cache is presented. The proposed analysis assumes a stationary access pattern, as shown in Fig. 1.10(a). First, the analysis for NEM is discussed in Section 3.5.1. Then, the analysis for TTL-IE and TTL-T is discussed in Sections 3.5.2 and 3.5.3, respectively.

3.5.1 LRU Cache under NEM

Similar to the analysis in Section 3.4.1 for FIFO under NEM, the Markov chain in Fig. 3.7 is used to model the state of object $i$ for LRU under NEM. The Markov chain in Fig. 3.7 has two types of states: NCS (state 0) and CFSs (states 1 to $C$).

In Fig. 3.7, the CFSs represent the relative time at which object $i$ was requested. State $C$ corresponds to the most recently requested object. The recency of a requested object decreases as the object goes from CFS $j$ to CFS $j-1$. State 1 corresponds to the least recently requested object.

In Fig. 3.7, an object $i$ moves from the current state $j \in [1, C-1]$ to state $C$ with rate $\lambda(i)$ (i.e. when object is requested). Also, object $i$ moves from state $C$ to state $C-1$ when any other object is requested (with rate $\beta - \lambda(i)$). Furthermore, an object $i$ moves from the current state $j$ to the next lower-numbered one when an object occupying a state $w \leq j$ is requested. Thus, the rate at which a cached
object $i$ moves from the current state to the next lower-numbered one, $\epsilon^z(t, i, j)$, can be calculated as

$$\epsilon^z(t, i, j) = \sum_{m=1, m \neq i}^{M} \lambda(m) \sum_{c=0}^{j} P^{z-1}(t, m, c)$$

(3.26)

Similar to FIFO under NEM, the analysis proposed in Table 3.1 is used for estimating the IHR of LRU under NEM, such that $\epsilon^z(t, i, j)$ is calculated in step 5 using (3.26).

3.5.2 LRU Cache under TTL-IE

Like the analysis for FIFO under TTL-IE, the analysis for LRU under TTL-IE accounts for the object expiry by introducing two main changes to the Markov chain shown in Fig. 3.7. First, the rate at which an object $i$ moves from the current CFS to NCS is increased by $\mu(i)$, as shown in Fig. 3.8. Second, an object $i$ moves from the current CFS $j < C$ to the next CFS $j + 1 \leq C$, if another object $m \neq i$ that currently occupies a cache state $c > j$ expires. Thus, an object $i$ moves from the current CFS to the next CFS with rate equal to $\gamma^z(t, i, j)$, which is calculated using (3.22), where
Note that, in Fig. 3.8, \( \epsilon^z(t, i, j) \) is calculated using (3.26).

Figure 3.8: LRU Markov chain for object \( i \) under TTL-IE.

Like FIFO under TTL-IE, the analysis shown in Table 3.1 can be used to calculate the IHR of LRU under TTL-IE, where \( \gamma^z(t, i, j) \) and \( \epsilon^z(t, i, j) \) are in step 5.

### 3.5.3 LRU Cache under TTL-T

Like FIFO under TTL-T, the analysis proposed in the previous section can be modified for LRU under TTL-T by associating a CES to each CFS, as shown in the Markov chain in Fig. 3.9.

In Fig. 3.9, the state \( 2C - 1 \) is occupied by the most recently requested object, if the object is fresh. Thus, when an object \( i \) is requested from any CFS, it moves to state \( 2C - 1 \) with rate \( \lambda(i) \). Also, when an object \( i \) is requested from any CES, it moves to state \( 2C - 1 \) with rate \( \varsigma(i) = \eta \lambda(i) \). Note that the state \( 2C \) is occupied by the most recently requested object, assuming that the object is expired. Thus, when an object \( i \) is requested from any CES, it moves to state \( 2C \) with rate \( \lambda_i - \varsigma(i) \).

Moreover, an object \( i \) in state \( 2C - 1 \), or state \( 2C \), moves to the previous state with rate \( \beta - \lambda(i) \) (i.e. if any object other than \( i \) is requested). An object \( i \) moves
from the current CFS $j$ to the associated CES $j + 1$, when it expires. Thus, an object $i$ moves from a CFS to the associated CES with rate $\mu(i)$.

An object $i$ moves from the current CFS $j$ to the previous CFS $j - 1$ (e.g. from state 3 to 1), or moves to the NCS (i.e. from state 1 to 0), when an object that occupies state $c < j$, is requested. Thus, object $i$ moves to the next lower CFS, or to the NCS, with a rate that is equal to $\epsilon^z(t, i, j)$, where $j$ is odd, such that

$$\epsilon^z(t, i, j) = \sum_{m=1, m \neq i}^{M} \lambda(m) \sum_{c=0}^{j+1} P^z(t, m, c) \quad \text{for odd } j \quad (3.27)$$

Similarly, object $i$ moves from the current CES to the next lower CES, or to the NCS, with rate $\epsilon^z(t, i, j)$, which is calculated using (3.26), where $j$ is even.

Similar to FIFO under TTL-T, the analysis proposed in Table 3.1 is used for estimating the IHR of LRU under TTL-T, where $S^z(t, i)$, $P^z(t, i, j)$, and $\nu^z(t)$ are calculated according to (3.23), (3.24), and (3.25), respectively.
3.6 Generalized Analysis of a Non-stationary Access Pattern

In the previous sections, the proposed analysis assumed a stationary access pattern, where the access pattern is fixed from the moment that an empty cache starts to operate until the cache reaches the SSHR, as described in Section 1.2. In this section, the extension of the proposed analysis for a non-stationary access pattern is presented. In a non-stationary access pattern, the access pattern characteristics may change after, or before, the cache reaches the SSHR, as shown in Figs. 1.10(b, c). Particularly, this section shows how the proposed analysis can be generalized for estimating the IHR, assuming new popular objects are periodically generated at the OS.

This section is organized as follows. The analysis of Infinite Cache under a non-stationary access pattern is proposed in Section 3.6.1. Then, the analysis of FIFO and LRU under a non-stationary access pattern is presented in Section 3.6.2.

3.6.1 Generalized Analysis of Infinite Cache

For Infinite Cache under a non-stationary access pattern, the IHR is calculated using (3.1) for NEM, TTL-IE, or TTL-T. However, the following modifications to the analysis presented in Section 3.2 are required:

1- Assuming that the number of objects changes at time $t_v$ to $M(t_v)$, if the number of objects remains fixed within the time interval $(t_v, t_1)$, then the probability matrix of object $i$ at time $t_1$, $P(t_1, i)$, is calculated by modifying (3.2) to

$$P(t_1, i) = \exp(Q(i) \Delta t)$$

(3.28)
where $\Delta t = t_1 - t_v$ and $t_1 > t_v$. Note that (3.28) is used for NEM, TTL-IE, and TTL-T.

2- $S(t_1, i)$ has to be calculated based on the probabilities $P(t_v, i, j), \forall i, j$. Note that $S(t_1, i)$ cannot be calculated based on the probabilities $P(t_0 = 0, i, j), \forall i, j$ (i.e. $S(t_1, i)$ cannot be calculated using (3.5)), if the access pattern is not stationary within the interval $(t_0, t_1)$, as shown in Fig. 3.10. Therefore, for NEM and TTL-IE, $S(t_1, i)$ is calculated by modifying (3.5) to (3.29). Similarly, for TTL-T, $S(t_1, i)$ is calculated by modifying (3.9) to (3.30).

\[
S(t_1, i) = \sum_{r=1}^{2} \left[ P(t_v, i) \right]_{r,1} P(t_v, i, r-1) \tag{3.29}
\]

\[
S(t_1, i) = \sum_{r=1}^{2} \left[ P(t_1, i) \right]_{r,1} P(t_v, i, r-1) + \eta \sum_{r=1}^{3} \left[ P(t_1, i) \right]_{r,3} P(t_v, i, r-1) \tag{3.30}
\]
3.6.2 Generalized Analysis of FIFO and LRU

Similar to Infinite Cache, $P^z(t_1, i)$ can be calculated for FIFO and LRU by modifying (3.19) to

$$P^z(t_1, i) = \exp(Q^z(i) \Delta t)$$

where $\Delta t = t_1 - t_v$. Then, $S^z(t_1, i)$ is calculated for NEM or TTL-IE by modifying (3.20) to

$$S^z(t_1, i) = \sum_{r=1}^{C+1} [P^z(t_1, i)]_{r,1} P^*(t_v, i, r - 1)$$

where the superscript $(.)^*$ indicates the final fixed point state probability solution provided by the contraction mapping analysis in Section 3.5. Similarly for TTL-T, $S(t_1, i)$ is calculated by modifying (3.23) to

$$S^z(t_1, i) = \sum_{r=1}^{2C+1} [P^z(t_1, i)]_{r,1} P^*(t_v, i, r - 1) + \eta \sum_{j=1}^{C} \sum_{r=1}^{2C+1} [P^z(t_1, i)]_{r,2j+1} P^*(t_v, i, r - 1)$$

3.7 FB-FIFO Replacement Policy

Unlike Perfect-LFU, LRU and FIFO do not store any information about the popularity of objects (i.e. number of requests), whether they are currently cached or not. Hence, LRU and FIFO can be easily implemented with a linked list, with length $C$. However, compared to FIFO, LRU requires more overhead on every cache hit to an object in order to change its state to $C$ (i.e. to move it to the front of the list as the most recently requested object) [46, 87, 148]. Many replacement policies
are introduced in order to enhance the SSHR of LRU, such as LRU-k, ARC, and LRFU [92, 98, 149]. However, these policies are even more complex than LRU, and they require additional resources.

In Section 3.4.1, the Markov chain representation of FIFO showed that the hit ratio degrades because FIFO allows all new objects, unpopular or not, to occupy state $C$. State $C$ is the position in the chain that allows the object to remain in the cache for the maximum amount of time. In this section, this insight is used to develop an improved version of FIFO policy, called frequency-based-FIFO (FB-FIFO).

The main idea behind FB-FIFO is to create a variable-sized protected segment in the cache ($S_p$) for objects that are not effectively one-timers. The object is effectively a one-timer if it cannot generate a cache hit, either due to its low request rate, or because it is requested only once throughout the cache operation.\footnote{A one-timer is an object that is requested once, and will never be requested again. The object is effectively a one-timer if it is requested once, or if it is requested more than once but the time intervals between requests are longer than the durations that the object lingers in the cache. Hence, the object that is effectively a one-timer cannot generate a cache hit.} The remainder of the cache is considered an unprotected segment ($S_u$). In FB-FIFO, it is assumed that both cache segments are managed separately with the FIFO policy. When an uncached object is requested, the object is moved to $S_u$ as the newest object. If $S_u$ is full, then the object that was brought into $S_u$ earliest will be ejected from the cache, as shown in Fig. 3.11.

If an object in $S_u$ experiences a cache hit (the object is not effectively a one-timer), the object is moved to $S_p$ as the newest object. If $S_p$ is full, the object that was brought into $S_p$ the earliest will move back to $S_u$ as the newest object. A
Figure 3.11: FB-FIFO: a new object enters the cache.

counter, \( n \), determines the capacity of \( S_p \), while the capacity of \( S_u \) is \( C - n \), such that \( 0 \leq n \leq N_{\text{max}} \), and \( N_{\text{max}} < C \). The initial value of \( n \) is set to zero at time \( t = 0 \). Every time an object in \( S_u \) experiences a cache hit, the value of \( n \) increments by one if \( n < N_{\text{max}} \). If \( n = N_{\text{max}} \), \( n \) cannot increment further, as shown in Fig. 3.12.

Note that, under TTL-IE, if an object in \( S_p \) expires, it will be ejected from the cache and the value of \( n \) decrements by one. However, under NEM or TTL-T, \( n \) does not decrement. Note that objects in \( S_p \) do not move back to \( S_u \) if \( n < N_{\text{max}} \). A flow chart of FB-FIFO is illustrated in Fig. 3.13.

As \( n \) increases, the probability that a new cached object lingers in \( S_u \) decreases. When a new object is cached in \( S_u \), it will be ejected if the next \( C - n \) requests are all cache misses. Therefore, as \( n \) increases, the probability that the objects that are moved to \( S_p \) (as the newest objects) are popular increases. Furthermore, the objects that are effectively one-timers will not affect the \( S_p \) queue. Hence, FB-FIFO is expected to outperform FIFO, which can easily be polluted by a sequence of requests for objects that are effectively one-timers.
FB-FIFO allows the current popular objects to be cached in $S_p$. In FB-FIFO, any object moves from $S_u$ to $S_p$ when it is requested within the next $C - n$ requests (excluding cache hits), regardless of the past average request rate of this object. Therefore, FB-FIFO is expected to adapt faster than Perfect-LFU to the changes in the popularity of the cached objects (i.e. FB-FIFO is more robust than Perfect-LFU), especially when the cached objects accumulate many requests over time. In this case, Perfect-LFU does not eject these cached objects, even if they are never requested.
Figure 3.13: FB-FIFO flow chart.

again. LFU-Aging [14] overcomes this problem by periodically reducing the counter values associated with cached objects.

The robustness of FB-FIFO may also decrease for small cache capacities, when \( n \) reaches \( N_{\text{max}} \) and does not decrease (i.e. under NEM and TTL-T, where \( n \) cannot decrease). If \( n = N_{\text{max}} = C - 1 \), only very popular objects can replace the past popular objects in \( S_p \), since a single object in \( S_u \) is ejected on the next cache miss. To overcome this problem, more complicated versions of FB-FIFO may be implemented to reset \( n \) to zero after a certain number of cache misses. In this chapter, the simplest version of the proposed FB-FIFO is evaluated, where \( n \) decreases under TTL-IE if and only if a cached object in \( S_p \) expires, while \( n \) does not decrease under NEM or TTL-T. Note that, under TTL-IE, as the object expiry rate increases, the robustness of FB-FIFO improves for small caches, since FB-FIFO under TTL-IE allows the expired
past popular objects in \( S_p \) to be ejected from the cache.

Note that determining a static value for \( n = N_{static} \) (i.e. fixed capacities for \( S_u \) and \( S_p \) throughout the evaluation interval) reduces the IHR of FB-FIFO. If \( N_{static} \) is increased, the speed at which \( S_p \) fills up decreases. Consequently, the IHR increases very slowly in the transient period starting from an empty cache. If \( N_{static} \) is decreased, the number of popular objects that can be stored in \( S_p \) decreases. Thus, the IHR for FB-FIFO decreases as \( N_{static} \) decreases, until FB-FIFO matches FIFO when \( N_{static} = 0 \).

3.8 Evaluation of the IHR under NEM and TTL-IE

An event-driven simulator was developed using C++ to verify the analytical results for Infinite Cache, LRU, and FIFO. Also, FB-FIFO and Perfect-LFU are implemented in the simulations to allow a comparison with LRU and FIFO. The workload generator is used to generate 1000 workload profiles that statistically conform with the proposed mathematical models in Section 3.1. For each simulated workload, the simulator determines the IHR at every simulation timestep, \( \delta t \). The final simulated IHR is then determined by averaging the IHR for all 1000 simulated profiles at each simulation timestep. For example, let \( H_{simulation}(t, n) \) denote the simulated IHR that is generated using the \( n \)th workload profile at time \( t \). The final simulated IHR using \( N \) workload profiles

\[ 4 \text{This large number of runs is required for achieving 95\% confidence interval with maximum error of estimate [71] that is lower than 0.3\%.} \]
profiles is calculated as

\[ H_{simulation}(t)|_N = \frac{1}{N} \sum_{n=1}^{N} H_{simulation}(t, n) \]  
\[ (3.31) \]

The IHR, \( H(t) \), is calculated over a duration of 3 to 20 hours in \( \delta t = 0.25 \) hour intervals as a function of cache capacity, object expiry rate, and request rate. Note that the analysis estimates the IHR at a specific time \( t \), while the simulated IHR using the \( n \)th profile, \( H_{simulation}(t, n) \), is the time domain average of the hit ratio over the interval \( \delta t \) (i.e. the number of cache hits divided by the total number of requests within \( \delta t \)), as shown in Fig. 3.14.

Figure 3.14: Simulated IHR is calculated at every simulation timestep, \( \delta t \).

In the analysis and simulations, the ratio between the cache size and the average object size (i.e. the cache capacity \( C \)) varies from 100 to 500. Though the analysis estimates the IHR assuming objects with different expiry rates, for simplicity, the IHR is evaluated in this chapter assuming that all the objects have the same expiry rate, \( \mu_e = \mu(i) \forall i \in [1, M] \). The request rates vary from 200 to 500 requests/hr, and are chosen such that the evolution of the IHR can be captured in the transient state.
The analysis and the simulator adopt the settings shown in Table 3.2. Note that the values in Table 3.2 are intended to illustrate the concepts in this chapter rather than representing empirical workload values.

Table 3.2: SCS: IHR evaluation factors and levels.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Objects</td>
<td>( M )</td>
<td>( 5000 - 10,000 )</td>
</tr>
<tr>
<td>Time Duration</td>
<td>( T )</td>
<td>( 3 - 20 ) hours</td>
</tr>
<tr>
<td>Simulation Timestep</td>
<td>( \delta t )</td>
<td>( 0.25 ) hour</td>
</tr>
<tr>
<td>Zipf Slope</td>
<td>( \alpha )</td>
<td>( 0.8 ) [11, 14]</td>
</tr>
<tr>
<td>Cache Capacity</td>
<td>( C )</td>
<td>( 100, 300, 500, ) Infinite Cache</td>
</tr>
<tr>
<td>Object Expiry Rate</td>
<td>( \mu_e )</td>
<td>( 0.5, 1 ) per hour [150]</td>
</tr>
<tr>
<td>Request Arrival Rate</td>
<td>( \beta )</td>
<td>( 200 - 500 ) per hour</td>
</tr>
</tbody>
</table>

In Section 3.8.1, the considered replacement policies are evaluated, assuming a stationary access pattern, with a fixed number of objects at the OS \( (M = 10,000) \). Section 3.8.2 illustrates the evaluation of replacement policies, assuming that the OS generates new popular objects periodically (i.e. a non-stationary access pattern is assumed). In the following, all plots show the hit ratio as a function of time (i.e. IHR). The markers in these plots represent the simulation results, while analysis results that correspond to each simulated scenario are shown with solid lines.

3.8.1 Evaluation of Replacement Policies under a Stationary Access Pattern

In this section, the IHR, \( H(t) \), is plotted within interval \([0, T]\) starting from an empty cache at time \( t = 0 \), where \( T = 210 \) minutes. Within this interval, it is assumed that the number of objects and the Zipf slope are fixed \((M = 10,000, \alpha = 0.8)\).

Fig. 3.15 shows a close match between analysis and simulation results for Infinite Cache, LRU, and FIFO assuming a cache capacity of \( C = 500 \), cached objects that
do not expire (i.e. $\mu_e = 0$), and request rate of $\beta = 500$. This plot also indicates that the hit ratio of the different replacement policies evolve differently as a function of time. In Fig. 3.15, Infinite Cache, LRU, FIFO, FB-FIFO, and Perfect-LFU have the same hit ratio until $t_{full} = 75$ minutes, where $t_{full}$ denotes the time when the cache becomes full. After $t_{full}$, the replacement policy starts replacing some cached objects, causing the hit ratio to increase in case of LRU, FB-FIFO, and Perfect-LFU, or decrease in case of FIFO.

Figure 3.15: IHR, where $C = 500$, $\mu_e = 0$, and $\beta = 500$.

Unlike the other replacement policies, Fig. 3.15 shows that FIFO reaches a maximum hit ratio of 31.4% during the transient period. Afterwards, FIFO decreases again and keeps fluctuating until reaching the SSHR of 29.4% after 210 minutes. Unlike the other replacement policies, FIFO does not exploit the popularity feature of objects. Thus, it is likely that some of the most popular objects that are cached within the first 30 minutes will reside at low cache states at $t_{full}$. Note that the hit ratio increases rapidly within the first 30 minutes when a big percentage of the most popular objects are cached. Hence, FIFO starts replacing a portion of the most popular objects with less popular ones after $t_{full}$, which causes the hit ratio to drop.
Afterwards, the hit ratio will increase again as FIFO starts caching the most popular objects, which could be replaced after $t_{\text{full}}$, again. An oscillation in hit ratio continues to be observed as the most popular objects work their way through the cache and get ejected. Eventually, steady state is reached once FIFO is able to maintain a fixed percentage of the most popular objects in the cache.

Fig. 3.15 shows that the proposed FB-FIFO outperforms both LRU and FIFO, while FB-FIFO achieves the same hit ratio as Perfect-LFU during the evaluation interval $[0, T]$. Note that Perfect-LFU and FB-FIFO start outperforming LRU at time $t = 120$ minutes. Before $t_{\text{full}}$, the three replacement policies maintain the same objects in the cache. However, unlike Perfect-LFU and FB-FIFO, LRU suffers from requests for objects that are effectively one-timers, which constrains the increase in the percentage of the cached popular objects after $t_{\text{full}}$.

The remainder of this section discusses the impact of cache capacity, object expiry, and request rate on the considered replacement policies.

**Impact of Cache Capacity**

Fig. 3.16 shows the impact of cache capacity, $C$, on $H(t)$ for the considered replacement policies. Fig. 3.16 shows that as $C$ decreases, the cache fills up quickly and thus, Perfect-LFU, FB-FIFO, and LRU start outperforming FIFO sooner.

Fig. 3.16(a) shows that FIFO fluctuates more rapidly and reaches steady state faster when $C$ decreases to 300. Table 3.3 helps illustrating the behavior for FIFO. Fig. 3.16(b) shows that when $C$ decreases to 100, FIFO fluctuation diminishes. When $C$ decreases, the percentage of the most popular objects that are cached when the
cache fills up at $t_{full}$ decreases. Therefore, after $t_{full}$, the probability that the portion of the popular objects that reside at low cache states will be replaced by objects with similar popularity increases. Consequently, the transient behavior of FIFO is smoothed faster as $C$ decreases.

**Table 3.3:** The IHR of FIFO, where $\mu_e = 0$ and $\beta = 500$.

<table>
<thead>
<tr>
<th>Time (minutes)</th>
<th>45</th>
<th>75</th>
<th>105</th>
<th>135</th>
<th>165</th>
<th>195</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C = 500$</td>
<td>25 %</td>
<td>31.4 %</td>
<td>28.3 %</td>
<td>28.9 %</td>
<td>30.4 %</td>
<td>28.8 %</td>
</tr>
<tr>
<td>$C = 300$</td>
<td>25 %</td>
<td>22.5 %</td>
<td>23.9 %</td>
<td>23.4 %</td>
<td>23.4 %</td>
<td>23.4 %</td>
</tr>
</tbody>
</table>

Fig. 3.16 shows that as $C$ decreases, FB-FIFO outperforms both LRU and FIFO more rapidly. Moreover, Perfect-LFU outperforms FB-FIFO after a time $t_p$ that
decreases with $C$, for example, $t_p = 120$ when $C = 300$, whereas $t_p$ decreases to 30 when $C = 100$. Note that if any replacement policy keeps a similar percentage of the popular objects as Perfect-LFU, that replacement policy achieves a hit ratio similar to Perfect-LFU. Therefore, after $t_{full}$, the difference between Perfect-LFU and FB-FIFO is small. This difference increases over time as Perfect-LFU identifies more of the popular objects. As $C$ increases, it takes longer for Perfect-LFU to distinguish popular objects with request rate that are too low to be identified by FB-FIFO. Thus, FB-FIFO tracks Perfect-LFU for a longer time.

**Impact of Object Expiry Rate**

Fig. 3.17 shows the impact of the object expiry rate, $\mu_e$, on $H(t)$. Fig. 3.17 shows that as $\mu_e$ increases, $H(t)$ decreases since the rate at which the object is ejected from the cache increases. Moreover, $H(t)$ reaches steady state faster as $\mu_e$ increases. Also, note that the transient behavior of FIFO diminishes as $\mu_e$ increases. As discussed in Section 3.8.1, this is due to the decrease in the percentage of the most popular objects that are cached when the cache fills up.

Let $H(t, 1)$ and $H(t, 2)$ denote the IHR in Fig. 3.17(a) and Fig. 3.17(b), respectively. Fig. 3.18 shows the relative change in the IHR (i.e. $(H(t, 1) - H(t, 2))/H(t, 1)$). In Fig. 3.18, the results show that Infinite Cache is the most sensitive to $\mu_e$, followed by Perfect-LFU, FB-FIFO, LRU, and FIFO. As $\mu_e$ increases, the popular objects are ejected due to object expiry, rather than requests for uncached objects. Thus, the replacement policies that minimize the ejection of popular objects are more sensitive to $\mu_e$. 

78
Impact of Request Rate

Figs. 3.19 and 3.17(a) show that, in the transient state, the IHR for all the replacement policies decreases when $\beta$ decreases to 250 requests/hr, while $\mu_e = 0.5$. This is consistent with the results for the SSHR of Infinite Cache generated in [11]. Moreover, comparing Figs. 3.19 and 3.17(a) shows that the relative increase in $\beta$ with respect to $\mu_e$ improves FB-FIFO more rapidly than LRU and FIFO, which improves the least.

Comparing Figs. 3.19 and 3.17(b) shows that doubling $\beta$ and $\mu_e$ for a replacement policy results in the same SSHR. However, the steady state is reached faster. For example, Perfect-LFU, FB-FIFO, and LRU reach steady state at time $t = 90$ when $\beta = 500$, while the time required to reach steady state is almost doubled when...
Figure 3.18: The relative change in the IHR when $\mu_e$ increases from 0.5 to 1, where $C = 300$ and $\beta = 500$.

Figure 3.19: IHR, where $C = 300$, $\mu_e = 0.5$, and $\beta = 250$.

$\beta = 250$.

3.8.2 Evaluation of Replacement Policies under a Non-stationary Access Pattern

In this section, the IHRs of the replacement policies are evaluated over $T = 20$ hours, assuming that the initial number of objects at time $t = 0$ is $M(t = 0) = 5000$. Every $\Delta T = 5$ hours, 50 new objects are generated. It is assumed that the 50 new generated objects become the most popular objects within the next $\Delta T$. For example, the most popular object within the interval $[0, 5]$ hours becomes the 50th most popular object within the interval $(5, 10]$ hours, and the 100th most popular object within the interval $(10, 15]$ hours. Similarly, the most popular object that is generated at $t = 5$ hours
becomes the 50th most popular object within the interval (10, 15] hours, and so on. Therefore, the initial popularity of the cached objects degrades over time.

Fig. 3.20 shows that the analysis accurately estimates the IHR of LRU and FIFO, assuming a non-stationary access pattern. Also, Fig. 3.20 illustrates the case when the cache does not reach steady state. Hence, the only way to estimate the hit ratio experienced by Web users is to calculate the IHR.

![Graph showing IHR comparison for Perfect-LFU, FB-FIFO, LRU, and FIFO with different parameters.]

(a) $\mu_e = 0$

(b) $\mu_e = 0.5$

Figure 3.20: IHR, where $C = 100$ and $\beta = 200$.

Fig. 3.20(a) shows that for $C = 100$, FB-FIFO adapts faster than the other replacement policies when new objects are generated at time $t = 5$ hours. Assume that the robustness of the replacement policy is denoted by the IHR achieved shortly after $M(t)$ changes (for example, $H(t = 5.5)$ after $M(t)$ changes at $t = 5$). Fig. 3.20(a)
shows that FB-FIFO is the most robust replacement policy after $t = 5$ hours. After $t = 10, 15$ hours, LRU becomes the most robust replacement policy followed by FIFO, FB-FIFO, and then Perfect-LFU. At $C = 100$, the popularity of the cached objects decreases greatly for Perfect-LFU and FB-FIFO every time 50 new popular objects are generated. Hence, Perfect-LFU and FB-FIFO degrade greatly over time since they take longer to eject formerly popular objects. As discussed in Section 3.7, after $t = 15$ hours, it might be useful for FB-FIFO to reset the counter to zero, or decrease its counter value on every cache miss, which may help remove old popular objects faster. Also, as discussed in Section 3.7, as $\mu_e$ increases, FB-FIFO becomes more robust, as shown in Fig. 3.20(b).

Note that the ratio between the number of new popular objects and the cache capacity plays a key role in the relative robustness of the replacement policies. Fig. 3.21(a) shows that as $C$ increases to 300, after $t = 10, 15$ hours, FB-FIFO becomes the most robust replacement policy followed by LRU, Perfect-LFU, and FIFO. The relative robustness of FB-FIFO increases each time new objects are generated. As discussed in Section 3.7, FB-FIFO caches the current popular objects in $S_p$ faster as $C$ increases, regardless of the past request rate of these objects.

Fig. 3.21(b) shows that as $C$ increases to 500, FB-FIFO matches Perfect-LFU for a longer time starting from an empty cache at time $t = 0$, as discussed in Section 3.8.1. Furthermore, Figs. 3.21(a, b) show that after $t = 15$ hours, FB-FIFO outperforms Perfect-LFU for a longer duration when $C$ increases from 300 to 500. As discussed in Section 3.7, Perfect-LFU is affected more than FB-FIFO when more cached objects accumulate many requests over time. In this case, Perfect-LFU takes longer than
FB-FIFO to identify the new popular objects and get rid of the old ones.

3.9 Evaluation of the SSHR under TTL-IE and TTL-T

This section evaluates the steady state hit ratio (SSHR) of FIFO, LRU, and FB-FIFO, assuming a stationary access pattern. The analysis and simulator adopt the settings shown in Table 3.4. The values in this table are meant to illustrate the concepts in this section, and they are not taken directly from empirical workload values.

In the following, all plots show the SSHR as a function of the cache capacity, $C$. Note that the markers in these plots represent the simulation results, while analysis results are shown with solid lines.
Table 3.4: SCS: SSHR evaluation factors and levels.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Objects</td>
<td>$M$</td>
<td>1000</td>
</tr>
<tr>
<td>Zipf Slope</td>
<td>$\alpha$</td>
<td>0.8 $[14,11]$</td>
</tr>
<tr>
<td>Aggregate Request Arrival Rate</td>
<td>$\beta$</td>
<td>200 per hour</td>
</tr>
<tr>
<td>Cache Capacity</td>
<td>$C$</td>
<td>20-100</td>
</tr>
<tr>
<td>Object Expiry Rate</td>
<td>$\mu_e$</td>
<td>10, 1 per hour $[150]$</td>
</tr>
<tr>
<td>Validation Factor</td>
<td>$\eta$</td>
<td>0.2, 0.5, 0.8</td>
</tr>
</tbody>
</table>

Fig. 3.22(a) shows the SSHR of FIFO at $\mu_e = 10$ (relatively high expiry rate $[150]$).

Fig. 3.22(a) shows that the SSHR for TTL-IE slightly outperforms the one for TTL-T when $C$ is small (e.g. 20) and $\eta$ is high (e.g. 0.8). Fig. 3.22(a) shows that as $C$ increases, or $\eta$ decreases, the SSHR for TTL-T improves faster than for TTL-IE. This is because as $C$ increases, TTL-IE does not make full use of the available cache capacity since the cached objects are ejected once they expire. On the other hand, TTL-T allows expired objects to stay in the cache, which results in an increased number of cache hits, especially when $\eta$ is small.

Fig. 3.22(b) shows the SSHR of FIFO at $\mu_e = 1$ (i.e. relatively moderate expiry rate). Fig. 3.22(b) shows that both TTL-T and TTL-IE yield similar SSHR when the cache is small, regardless of $\eta$. This is because ejections due to caching new objects dominate ejections due to object expiry in TTL-IE. Fig. 3.22(b) shows that as $C$ increases, the SSHR for TTL-IE outperforms the one for TTL-T even for a low $\eta$ (e.g. 0.2). This is because TTL-IE allows the cache to eject unpopular objects from the FIFO queue faster, while TTL-T wastes the cache capacity by keeping the expired unpopular objects in their regular order in the FIFO queue.

Fig. 3.23 shows that the SSHR of LRU acts like FIFO for both TTL implement-
Figure 3.22: SSHR of FIFO.

The only difference appears in Fig. 3.23(b) at $\mu_e = 1$ and $\eta = 0.2$, where the SSHR of LRU under TTL-T matches the SSHR of LRU under TTL-IE, when $C < 80$. Also, for LRU, the SSHR under TTL-T is better than the SSHR under TTL-IE, when $C \geq 80$. This is because, under TTL-T, LRU allows unpopular objects to be ejected faster than FIFO, since LRU reorders the cache queue according to recency (or popularity, since a stationary access pattern is assumed [11]).

Fig. 3.24 shows that FB-FIFO outperforms LRU and FIFO under TTL-T when the cache is small, as suggested by the results in Section 3.8.1. Fig. 3.24 shows that the SSHR of FB-FIFO degrades rapidly under TTL-IE as $\mu_e$ increases. This is consistent with the results in Section 3.8.1.
Figure 3.23: SSHR of LRU.

Under TTL-T, Figs. 3.23 and 3.24 show that, compared to the SSHR of LRU, the SSHR of FB-FIFO is very sensitive to the value of $\eta$. For example, Figs. 3.23(a) and 3.24(a) show that when $\mu_e = 10$ and $C = 20$, the SSHR of LRU increases by 3%, in case of LRU, and increases by 10%, in case of FB-FIFO, when $\eta$ decreases from 0.8 to 0.2.

Moreover, Fig. 3.24(b) shows that the sensitivity of FB-FIFO to $\eta$ decreases more rapidly than LRU as $\mu_e$ decreases (i.e. as the probability of objects being in a CES decreases). For example, Figs. 3.23(b) and 3.24(b) show that when $\mu_e = 1$ and $C = 20$, the SSHR of LRU increases by 0.3%, in case of LRU (i.e. the sensitivity to $\eta$ decreases by 2.7% compared to when $\mu_e = 10$), and increases by 3%, in case of

86
FB-FIFO (i.e. the sensitivity to $\eta$ decreases by 7% compared to when $\mu_e = 10$), when $\eta$ decreases from 0.8 to 0.2.

![Graph showing SSHR of FB-FIFO for different cache capacities and hit ratios for TTL-IE and TTL-T, with $\eta$ values of 0.2, 0.5, and 0.8.](image)

(a) $\mu_e = 10$

(b) $\mu_e = 1$

Figure 3.24: SSHR of FB-FIFO.

Note that, under TTL-T, FB-FIFO protects the popular objects from being ejected (i.e. FB-FIFO cached popular objects in the protected segment $S_p$), even after they expire, as discussed in Section 3.7. Hence, the SSHR of FB-FIFO is very sensitive to the probability that the expired protected objects are invalid, $\eta$, as well as the object expiry rate, $\mu_e$. On the other hand, compared to FB-FIFO, LRU allows more expired popular objects to be ejected, which is why LRU is less sensitive to $\eta$ and $\mu_e$.  

87
3.10 Summary

In this Chapter, the proposed analytical models for Infinite Cache, Perfect-LFU, FIFO, and LRU were presented. Then, a new replacement policy, FB-FIFO, was introduced. In Section 3.8, the replacement policies were evaluated under NEM and TTL-IE. The results in Section 3.8 showed that different replacement policies have very different transient performance, whether the popularity of cached objects is fixed or not. This makes the proposed analysis in this chapter valuable when considering scenarios like changing popularity or mobile caching. The following list summarizes the major findings in Section 3.8:

- For a stationary access pattern, where the number of objects and the object popularity are fixed during the transient period:
  1. Unlike the other replacement policies studied in this chapter, FIFO reaches its maximum hit ratio during the transient fluctuations that occur before reaching steady state. These fluctuations diminish as the cache capacity decreases, or as the object expiry rate increases with respect to the request rate.
  2. FB-FIFO outperforms both LRU and FIFO, especially when cache capacity is small. Moreover, FB-FIFO matches Perfect-LFU for a longer time as the cache capacity increases.
  3. As the object expiry rate increases with respect to the request rate, the IHR of Infinite Cache decreases more rapidly than Perfect-LFU, FB-FIFO, LRU, and FIFO, which decreases the least.

- For a non-stationary access pattern, where new popular objects are periodically
1- FB-FIFO outperforms LRU and FIFO and adapts faster than the other replacement policies to the changes in the popularity of the cached objects (i.e. FB-FIFO is more robust) when the cache capacity is large relative to the number of newly generated objects.

2- For small caches under TTL-IE, the robustness of FB-FIFO improves as the object expiry rate increases.

Furthermore, in Section 3.9, the SSHRs of FIFO, LRU, and FB-FIFO were evaluated under a stationary access pattern. The results for FIFO, LRU, and FB-FIFO showed that the SSHR under TTL-T and TTL-IE become closer when the cache size or the object expiry rate decreases. For small caches, it is the replacement policy rather than the consistency mechanism that plays the key role in enhancing the SSHR, if the object expiry rate is small. This is consistent with the findings in [110].

For large caches, the results showed that the SSHR for TTL-IE outperforms the one for TTL-T, when the object expiry rate is moderate (e.g. 1 per hour) and \( \eta \) is large (i.e. the probability that the validation request is served from the OS is large). However, when the object expiry rate is high (e.g. 10 per hour), the SSHR under TTL-T outperforms the one under TTL-IE. Also, the SSHR under TTL-T is very sensitive to the value of \( \eta \), especially for large caches or when the object expiry rate is high.
Chapter 4

Analysis of the Hierarchical Cache System (HCS)

This chapter presents an analytical model for estimating the instantaneous hit ratio (IHR) and the instantaneous average hit distance (IAHD) for a two-level hierarchical cache system. The analytical model in this chapter is based on the analysis of the SCS introduced in Chapter 3.

In this chapter, two HCSs are considered. The first HCS is shown in Fig. 1.3, where it is assumed that the first level has many fixed leaf caches (FLCs), while the second level has one fixed root cache. Note that a fixed cache refers to the cache that is always connected to the network and is available to serve objects. The second HCS considers the case where the leaf caches are hosted by users that disconnect from the network at random, as discussed in Section 1.1.4. These will be called temporary leaf caches (TLCs). In this scenario, it is assumed that some users in the network have extra power and cache storage to serve other users. However, a TLC is available to the other users in the network only when the user hosting this TLC is connected to the network, as will be discussed in detail in Section 4.4.

The analysis presented in this chapter determines both the IHR and the IAHD for the proposed HCS, which uses leave copy everywhere (LCE) as a sharing protocol and TTL-T or TTL-IE as a consistency mechanism. This is under the assumption that each cache in the cache hierarchy independently applies the LRU replacement policy. Note that the analysis introduced in this chapter is also applicable for Infinite
Cache and FIFO. However, this chapter merely considers LRU for conciseness.

Moreover, this chapter introduces a new sharing protocol, called promote cached objects (PCO). The analysis for estimating both the IHR and the IAHD of the HCS using PCO under TTL-IE is also provided. Both analysis and simulation will show that PCO is very robust when the object access pattern changes due to the generation of new popular objects at the OS. Under the assumption of a stationary access pattern, it will also be shown that PCO outperforms both LCE and LCD in terms of steady state average hit distance (SSAHD), especially under TTL-T.

The remainder of this chapter is organized as follows. First, the proposed cache performance metrics and the analysis assumptions are discussed in Section 4.1. Second, the performance analysis of LCE is presented in Section 4.2. Next, the design and analysis of PCO are presented in Section 4.3. After that, the analysis for TLCs is discussed in Section 4.4. Then, cache performance results generated using both analysis and simulation are presented in Section 4.5. Finally, concluding remarks are made in Section 4.6.

4.1 Evaluation Metrics and Assumptions

During the transient periods, the IHR of the SCS is the only way to evaluate the user’s perceived latency, as discussed in Chapter 3. Moreover, for the HCS, the user’s perceived latency at time $t$ can be characterized by evaluating the performance of the cache hierarchy in terms of the instantaneous average hit distance (IAHD) (i.e. the average number of links traversed to retrieve the requested object at time $t$), as
discussed in Section 1.2. The IAHD of the HCS, $D_U(t)$, can be calculated as

\[
D_U(t) = D_L H_L(t) + D_R H_R(t) + D_O (1 - H_L(t) - H_R(t))
\]  

(4.1)

where $D_L$, $D_R$, and $D_O$ are constants that represent the number of links between the user and the leaf cache, the root cache, and the OS, respectively. Also, $H_L(t)$ and $H_R(t)$ are the IHRs of the leaf cache and the root cache, respectively. In order to calculate $D_U(t)$, analytical models for calculating $H_L(t)$ and $H_R(t)$ for LCE and PCO are proposed in Section 4.2 and Section 4.3, respectively.

The analysis in this chapter adopts the access pattern characteristics introduced for the SCS in Section 3.1. Hence, the users’ requests for $M(t)$ objects at time $t$ follow a Poisson process with rate $\beta$, and the initial TTL value of a cached object $i$ is exponentially distributed, with mean $1/\mu(i)$. Moreover, the analysis in this chapter assumes homogenous populations are connected to $K$ leaf caches. Thus, all leaf caches experience the same request rate, $\beta_L = \beta/K$, with the same Zipf slope, $\alpha$. Note that the proposed analysis can be easily extended for heterogeneous populations, and homogenous populations are merely assumed for conciseness. Also, for simplicity, it is assumed that the objects are of the same size, and the all leaf caches have the same size. Thus, the leaf cache and root cache can store up to $C_L$ and $C_R$ objects, respectively.

\[\text{1The steady state performance of a hierarchical cache was evaluated in [41,70] in terms of the average hit distance.}\]
4.2 Analysis of LCE

Assuming fixed leaf caches (FLCs), the IHR of a leaf cache, $H_L(t)$, can be directly characterized using the SCS analysis introduced in Section 3.5. Moreover, this section shows how the analysis in Section 3.5 can be extended for the LCE root cache, assuming FLCs.

Under NEM, TTL-IE, or TTL-T, the Markov chain analysis in Section 3.5 can be used to calculate the IHR of the root cache, such that (3.21) is modified to

$$H_R^z(t) = \sum_{i=1}^{M} \frac{1 - S_R^z(t, i)}{\sigma i^\alpha} S_L^*(t, i)$$  \hspace{1cm} (4.2)

where $S_L^*(t, i)$ represents the probability that object $i$ is not served by the leaf cache at time $t$. Note that the superscript $\cdot^*$ indicates the final fixed point state probability solution provided by the contraction mapping analysis in Section 3.5. Note that $S_R^z(t, i)$ denotes the probability that object $i$ is not served by the root cache at time $t$, and $H_R^z(t)$ is the IHR of the root cache. Also, the superscript $\cdot^z$ denotes the iteration number in the proposed contraction mapping analysis, while $\sigma = \sum_{i=1}^{M} \frac{1}{i^\alpha}$ and $\alpha \in [0, 1]$ is the Zipf slope.

Note that, in this chapter, (4.2) is used assuming that the miss events at the leaf cache and the hit event at the root cache are statistically independent. This assumption holds when many leaf caches are assumed (i.e. when $K$ is large) [41,151]. For a small $K$, (4.2) generates optimistic results, as will be discussed in detail in Appendix B.

Moreover, if the leaf caches are in a transient period, the analysis must account for
the non-stationary access pattern experienced by the root cache. If the leaf caches are in a transient period, then the root cache experiences a non-stationary request rate that depends on the IHR of the leaf caches. At the root cache, the average request rate for object $i$ at time $t$ is

$$\lambda_R(t, i) = K \lambda_L(i) S_L^*(t, i)$$

(4.3)

where the average request rate of the Poisson process for object $i$ at a leaf cache, $\lambda_L(i)$, is equal to

$$\lambda_L(i) = \frac{\beta_L}{\sigma i^\alpha}$$

(4.4)

As discussed in Section 3.6, the IHR of the leaf cache in a transient period, $H_L^*(t)$, can be calculated assuming a stationary access pattern within the time interval $(t_0, t)$, where the probabilities $P_L^*(t_0, i, j), \forall i, j$ of the leaf cache at time $t_0$ are known, regardless of the value of $\Delta t = t - t_0$. However, within the time interval $(t_0, t)$, the root cache experiences a non-stationary access pattern, due to non-stationary $\lambda_R(t, i)$, according to (4.3). Therefore, $H_R^*(t)$ cannot be calculated based on the probabilities $P_R^*(t_0, i, j), \forall i, j$ of the root cache at time $t_0$, especially when $\Delta t$ is large.

In order to calculate $H_R^*(t)$, the interval $(t_0, t)$ is divided into small increments of $\Delta t$. Then, the probabilities $P_R^*(t, i, j), \forall i, j$ are calculated at each of these time increments using the single cache analysis in Section 3.5. Then, $H_R^*(t)$ is calculated based on the probabilities $P_R^* (t - \Delta t, i, j), \forall i, j$, rather than $P_R^*(t_0, i, j), \forall i, j$. For example, Fig. 4.1 illustrates the process of calculating $H_R^*(t = 3\Delta t)$, assuming that the
access pattern at the leaf caches is stationary within the intervals $(t_0, 3\Delta t)$. In Fig. 4.1, it is assumed that the leaf caches are in a transient period. Note that calculating $P^*_R(t, i, j)$ requires the value of $\lambda_R(t, i)$, which depends on $S^*_L(t, i)$ according to (4.3). Hence, the values of $S^*_L(t, i) \forall i$ also have to be calculated every $\Delta t$.

4.3 Promote Cached Object (PCO)

In this section, the motivation for developing PCO is discussed in Section 4.3.1. Then, the PCO protocol is introduced in Section 4.3.2. Finally, the analysis of PCO under TTL-IE is provided in Section 4.3.3.

4.3.1 PCO Motivation

The drawback of LCE is that it stores multiple copies of a new object in the cache hierarchy, even if it is effectively a one-timer\(^2\). This allows the cache hierarchy to be

\(^2\)An object is effectively a one-timer if its request rate is too low to generate a cache hit, as discussed in Section 3.7.
easily polluted with unpopular objects.

In [25], the SSHR is improved by allowing the cache hierarchy to store only the popular objects, such that the leaf caches are preserved for the most popular objects. In [25], the objects that are effectively one-timers are not cached in any cache throughout the cache hierarchy. As discussed in Section 2.5.1, Laoutaris et al. [24] proposed a simpler protocol, called LCD, to improve the SSAHD for the leaf caches by preventing the leaf caches from caching the objects that are effectively one-timers.

However, in steady state, LCD yields leaf caches with a similar set of the most popular objects (assuming homogenous users and leaf caches). Since the root cache is usually larger than the leaf cache [25], it is better to have a larger set of popular objects in the root cache than a smaller set of popular objects in the leaf caches. Hence, the proposed PCO protocol uses the root cache, rather than the leaf caches, to store popular objects. This considerably improves the SSAHD experienced by the users, as will be shown in Section 4.5.1.

Compared to LCE, using the techniques in [24,25] or PCO might decrease the robustness of the cache hierarchy under a non-stationary access pattern. This is because more time will be required to detect and store the newly generated popular objects, either in the leaf caches, in the case of [24,25], or in the root, in the case of [25] and PCO. The robustness of LCE, LCD, and PCO under these conditions is evaluated in detail in Section 4.5.2.
4.3.2 PCO Protocol under TTL-IE or TTL-T

In PCO, rather than leaving a copy of an object in the root cache as it travels down the hierarchy from the OS, PCO stores an object in the root, only if the object generates a cache hit at a leaf cache (i.e. the object is not effectively a one-timer). Under TTL-IE or TTL-T, if a user accesses an object at the leaf cache, the leaf cache tells the root cache to download this object from the OS, if it does not already have it. Note that in DEMOTE [119] and the protocol proposed in [25], the object is sent to the root cache when it is ejected from a leaf cache, while in PCO the object is sent to the root cache when it experiences a cache hit at a leaf cache.

Note that it may take time to transfer the object from the OS to the root cache. However, it is assumed that the root cache can serve this object in the next request as suggested by [25,119]. This is a realistic assumption when the root cache operates in cut-through mode (when a cache begins to receive an object, it is able to immediately forward the object to the leaf cache while the object is being received) [7].

Under TTL-IE, PCO and LCE achieve the same IHR for leaf caches. However, since PCO preserves the root cache for objects that are not effectively one-timers, PCO improves the IHR of the root cache, as will be shown in Section 4.5. In the following, the analysis for estimating the IHR of PCO under TTL-IE is presented.

4.3.3 Analysis of PCO under TTL-IE

As discussed in Section 4.3.2, the leaf cache achieves the same SSHR for LCE and PCO. Hence, like LCE, the analysis introduced in Section 3.5 can be used for PCO, assuming FLC. Also, the Markov chain analysis introduced in Section 4.2 can be
Figure 4.2: LRU Markov chain for object $i$ in PCO root cache under TTL-IE.

used to calculate the IHR of the PCO root cache. However, two modifications are required, as shown in Fig. 4.2. First, since the root cache is preserved for objects that are promoted (generate a hit) by a leaf cache, the rate at which object $i$ moves from state 0 to state $C_R$ is modified to

$$
\varphi_R(t, i) = K\lambda_L(i)(1 - S^*_L(t, i))
$$

(4.5)

Note that, in LCE, $\varphi_R(t, i) = \lambda_R(t, i)$ but this is not true for PCO. Second, as discussed in Sections 3.5, an object $i$ moves from the current CFS $j$ to the next lower CFS $j - 1$ when any new object $m \neq i$ is cached, or due to a cache hit to an object $m \neq i$ that occupies a cache state below $j + 1$. Thus, for the PCO root cache, the rate at which an object $i$ moves from the CFS $j$ to the CFS $j - 1$ is

$$
\epsilon^*_R(t, i, j) = \sum_{m=1, m \neq i}^M \varphi_R(t, m)P^z_{R}(t,m,0) + \sum_{m=1, m \neq i}^M \lambda_R(t, m)\sum_{c=1}^{j} P^z_{R}(t,m,c)
$$

(4.6)

where $j \leq C_R$ and $\lambda_R(t, i)$ is calculated using (4.3). Note that (3.26) is a special case of (4.6), where $\varphi_R(t, m) = \lambda_R(t, m) \forall m$. 

98
4.4 Analysis of Temporary Leaf Caches (TLCs)

This section introduces an analytical model for estimating the IHR for TLCs running LRU under a stationary access pattern. First, Section 4.4.1 presents the TLCs network model and the assumptions. Then, in Section 4.4.2, the TLCs analysis is introduced.

4.4.1 TLCs Network Model and Assumptions

The proposed network model considers a WLAN, where some mobile users that have extra power and cache storage host TLCs. As discussed in Section 1.1.4, a TLC is available to other users in the WLAN only when the user hosting the TLC is connected to that WLAN. For the HCS with TLCs, it is assumed that each TLC is associated with a group of users who query this TLC, rather than the access point, as shown in Fig. 4.3. In Fig. 4.3, when a TLC disconnects from the HCS, its users will automatically be associated with the closest available TLC. When a new TLC joins the HCS, nearby users will automatically be associated with this new TLC. Note that it is assumed that the new TLC is empty when it joins the network [138].

As discussed in Section 1.1.4, the user’s request is satisfied from the associated TLC if the user’s TLC has the requested object. Otherwise, the user downloads the requested object via an access point. When the user’s TLC does not have the requested object, the user either communicates directly with the access point, or through the TLC using multihop communication [35, 71]. In both cases, the access

3Users automatically recognize the closest available TLC. Network management and configuration, as well as the frequency planning, are beyond the scope of this dissertation.
point forwards a copy of the requested object to the TLC.

Figure 4.3: The proposed TLCs network model.

The analysis in this section adopts the same connectivity model suggested in [51, 125, 152], where the TLCs population is modeled as a birth-death process (i.e. M/M/1 queue with Serve In Random Order (SIRO) discipline) [146], with an arrival rate that equals $\lambda_u$ and a departure rate that equals $\mu_u$, as shown in Fig. 4.4. The advantage of this birth-death assumption is that it allows a tractable analysis. Also, for VANET, the birth-death assumption agrees with the observation made in [51] that the number of vehicles passing the observer, who stands at an arbitrary point at a highway, per unit of time is a Poisson process. While it is one possible way to model the TLCs population, other models may of course offer better accuracy depending on the specific
user scenario being considered. From Fig. 4.4, the flow matrix for the TLCs is given by

\[
Q_u = \begin{bmatrix}
-\lambda_u & \lambda_u & 0 & 0 & \cdots \\
\mu_u & -(\mu_u + \lambda_u) & \lambda_u & 0 & \cdots \\
0 & \mu_u & -(\mu_u + \lambda_u) & \lambda_u & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\end{bmatrix}
\]  

(4.7)

Figure 4.4: Birth-death Markov chain model of the TLCs population.

Let \( U_{K(t)} \) denote the probability that the number of TLCs is \( K(t) \), which corresponds to the probability that the Markov chain representing the TLCs population is in state \( K(t) \) at time \( t \). This probability is equal to

\[
U_{K(t)} = [\exp(Q_u t)]_{1,K(t)+1}
\]  

(4.8)

In the steady state, \( \lim_{t \to \infty} U_{K(t)} \) can be calculated as

\[
\lim_{t \to \infty} U_{K(t)} = \rho^{K(t)} (1 - \rho)
\]  

(4.9)

where \( \rho = \frac{\lambda_u}{\mu_u} \). Note that when the population reaches steady state, the average number of TLCs becomes \( \frac{\lambda_u}{\mu_u - \lambda_u} \).
4.4.2 TLCs Analysis

The single LRU cache analysis in Section 3.5 can be extended for estimating the IHR of a TLC. This requires repeating the analysis introduced in Section 3.5 for each \( K(t) \), where \( K(t) = 1, 2, 3, \ldots, K_{\text{max}} \), and \( K_{\text{max}} \) is large. Hence, assuming a stationary access pattern, the IHR of the leaf cache at the \( z \)th iteration for a given \( K(t) \) is calculated after modifying (3.21) to

\[
H^z_{L}(t)|_{K(t)} = \sum_{i=1}^{M} \frac{1 - S^z_{L}(i)|_{K(t)}}{\sigma i^\alpha} \quad (4.10)
\]

where \( S^z_{L}(i)|_{K(t)} \) is the probability that object \( i \) is not served by a TLC at time \( t \), given a \( K(t) \), which can be calculated by modifying (3.20) to

\[
S^z_{L}(t, i)|_{K(t)} = \exp(Q^z_{L}(t, i)|_{K(t)} t) \quad (4.11)
\]

Since \( K(t) \) is a random quantity, \( H^z_{L}(t) \) can be determined by averaging (4.10) over the probabilities in (4.8) for \( K(t) = 0, 1, \ldots \) such that

\[
H^z_{L}(t) = \sum_{k=0}^{\infty} H^z_{L}(t)|_{K(t)} U_{K(t)} \quad (4.12)
\]

Therefore, \( H^z_{L}(t)|_{K(t)} \forall k \in [1, K_{\text{max}}] \) can be calculated using the analysis in Section 3.5. Then, (4.12) is used to calculate \( H^z_{L}(t) \), as illustrated in Table 4.1. However, in order to calculate \( H^z_{L}(t)|_{K(t)} \), the analysis in this section has also to account for the TLC disconnections as follows.

For \( K(t) \) TLCs under TTL-IE, the cached object is ejected either when it expires, or when the TLC caching the object leaves the network. Since the object is equally
Table 4.1: Iterative analysis for estimating the IHR of a TLC running LRU under TTL-IE.

1- Set: $z = 0$
2- Set: $P_L^0(t, i, 0) = 1 \forall i \in [1, M]$
3- Do
4- $z = z + 1$
5- Set: $K(t) = 0$
6- Do
7- $K(t) = K(t) + 1$
8- Calculate: $\gamma^z_L(t, i, j)|_{K(t)} \forall i \in [1, M]$ using (4.13)
9- Calculate: $\gamma^z_L(t, i, j)|_{K(t)} \forall i \in [1, M]$ using (4.14)
10- Calculate: $S^z_L(t, i)|_{K(t)} \forall i \in [1, M]$ using (4.11)
11- Calculate: $H^z_L(t)|_{K(t)}$ using (4.10)
12- While $(K(t) < K_{\text{max}})$
13- Calculate: $H^z_L(t)$ using (4.12)
14- While $(H^z_L(t) \neq H^{z-1}_L(t))$
15- Output $H_L(t) = H^*_L(t) = H^z_L(t)$

likely to be in any of the $K(t)$ caches, the probability that a departing TLC is taking this cached object is equal to $1/K(t)$. Thus, the rate at which an object $i$ returns to state 0 at time $t$ due to a TLC disconnection is $\xi(t)|_{K(t)} = \mu_u/K(t)$.

Since the minimum of two exponential random variables is an exponential random variable with rate equal to the sum of the rates of these two random variables, the rate at which the object moves from a cache-fresh state (CFS) to state 0 is equal to $\mu(i) + \xi(t)|_{K(t)}$. Note that $\gamma^z_L(t, i, j)$ and $\epsilon^z_L(t, i, j)$ are calculated for a given $K(t)$. Thus, (3.22) and (3.26) are rewritten as

$$
\gamma^z_L(t, i, j)|_{K(t)} = \sum_{m=1, m \neq i}^{M} \mu(m) + \xi(t)|_{K(t)} \sum_{c=j+1}^{C_i} P^z_{L}^{c-1}(t, m, c)|_{K(t)}
$$

(4.13)
\[ \epsilon^z_L(t, i, j)|_{K(t)} = \sum_{m=1, m \neq i}^{M} \lambda_L(t, m) \sum_{c=0}^{j} P^z_{L}^{-1}(t, m, c)|_{K(t)} \] (4.14)

where \( \lambda_L(t, i) = \frac{\beta \sqrt{t} \sigma^2}{\sigma^2} \), and \( \beta \) is the total request rate by all users in the HCS. It is important to note that, unlike the FLCs, the TLC always experiences a non-stationary access pattern, since \( \lambda_L(t, i) \) depends on the number of available TLCs, \( K(t) \). Hence, in order to calculate the \( H^*_L(t)|_{K(t)} \), the approach introduced in Section 4.2 for a cache under a non-stationary access pattern has to be used.

Furthermore, the analysis in this section can also be used for a TLC under TTL-T. However, \( \xi^z(t)|_{K(t)} \) is added to the paths from every state \( 0 < j \leq 2C \) to state 0 in the Markov chain in Fig. 3.9. Also, the objects move from the current state \( j \) to the state \( j + 2 \) with rate

\[ \gamma^z_L(t, i, j)|_{K(t)} = \xi(t)|_{K(t)} \sum_{m=1, m \neq i}^{M} \sum_{c=w}^{2C_L} P^z_{L}^{-1}(t, m, c)|_{K(t)} \] (4.15)

where \( w = j + 1 \), if \( j \) is even, and \( w = j + 2 \), if \( j \) is odd.

4.5 Cache Performance Evaluation

A synthetic workload generator and an event-driven simulator were developed using C++ to verify the results of the proposed analysis for LCE and PCO. Moreover, LCD is also implemented in the simulations to allow a comparison with LCE and PCO. The workload generator is used to generate 30 workload profiles\(^4\) that statistically conform with the proposed mathematical models in Section 4.1. For each simulated workload, \(\footnote{This achieves 95\% confidence interval with maximum error of estimate that is lower than 0.8\%.} \)

\[^4\text{This achieves 95\% confidence interval with maximum error of estimate that is lower than 0.8\%.}\]
the simulator determines the IHR of each cache in the HCS at every simulation timestep, $\delta t$. As discussed in Section 3.8, the final simulated IHR is then determined by averaging the IHR for all 30 simulated profiles at each simulation timestep. The final simulated IAHD at time $t$ is calculated using the simulated IHR values in (4.1).

Though the proposed analysis allows estimating the IAHD assuming objects with different expiry rates, for simplicity, the IAHD is evaluated assuming that all the objects have the same expiry rate $\mu_e$ (i.e. $\mu_e = \mu(i) \forall i \in [1, M]$).

The analysis and simulator adopt the settings shown in Table 4.2. Note that the values in Table 4.2 are meant to illustrate the concepts in this chapter rather than representing empirical workload values.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Objects</td>
<td>$M$</td>
<td>5000</td>
</tr>
<tr>
<td>Simulation Timestep</td>
<td>$\delta t$</td>
<td>15 minutes</td>
</tr>
<tr>
<td>Number of leaf caches</td>
<td>$K$</td>
<td>20</td>
</tr>
<tr>
<td>Zipf Slope</td>
<td>$\alpha$</td>
<td>0.8 [6, 14]</td>
</tr>
<tr>
<td>Request Rate</td>
<td>$\beta$</td>
<td>2000 per hour</td>
</tr>
<tr>
<td>Leaf Cache Capacity</td>
<td>$C_L$</td>
<td>50</td>
</tr>
<tr>
<td>Root Cache Capacity</td>
<td>$C_R$</td>
<td>50 - 250</td>
</tr>
<tr>
<td>Object Expiry Rate</td>
<td>$\mu_e$</td>
<td>0, 1 per hour [11, 150]</td>
</tr>
<tr>
<td>Validation factor</td>
<td>$\eta$</td>
<td>0.2, 0.8</td>
</tr>
</tbody>
</table>

In Table 4.2, it is assumed that the validation factor, $\eta$, equals 0.2 or 0.8. Note that $\eta$ is the probability that the expired cached object is invalid, and thus, cannot be served from the cache (i.e. the request for this expired object is counted as a cache miss, as discussed in Section 1.1.1). Hence, low $\eta$ implies that the OS generates very conservative TTL values. Also, note that the study in [75] suggested that $\eta$ should be as low as $\eta = 0.15$. However, it is interesting to evaluate the considered sharing
protocols using a larger $\eta$.

The following sections evaluate the instantaneous and the steady state hit ratio (IHR and SSHR) for LCE, PCO and LCD at the root cache (IHR-R and SSHR-R) and at the leaf caches (IHR-L and SSHR-L). Also, the instantaneous and steady state average hit distance (IAHD and SSAHD) will be illustrated, such that $D_L = 0$, $D_R = 2$ and $D_0 = 10 \ [70,111]$. In all the plots in the following sections, the data points are generated using simulation and the solid lines are generated using analysis.

The remainder of this section is organized as follows. Section 4.5.1 illustrates the SSAHD of the considered sharing protocols under NEM and TTL-IE. Then, Section 4.5.2 illustrates the SSAHD of the considered sharing protocols under TTL-T. Section 4.5.3 presents the evaluation of the IAHD of the considered sharing protocols under NEM, assuming a non-stationary access pattern. Finally, Section 4.5.4 studies the impact of TLCs on the SSAHD of the HCS, under NEM and TTL-IE.

4.5.1 Evaluation of the SSAHD under NEM and TTL-IE

The excellent agreement between the analytical curves and the simulated data points in Fig. 4.5 shows that the analysis accurately estimates the SSAHD for LCE and PCO as a function of $\mu_e$ and $C_R$.

Fig. 4.5(a) shows that, when $\mu_e = 0$, LCD achieves better SSAHD than LCE as a direct result of the rapid increase of the SSHR-L for LCD. This is consistent with the results in [24]. Unlike LCE, LCD prevents objects that are effectively one-timers from being stored in the leaf caches, which improves the SSHR-L, as discussed in Section 4.3.1. Note that the root cache for both LCE and LCD must have the same
objects. However, Fig. 4.5(a) shows that LCE achieves better SSHR-R because LCD satisfies more requests from the leaf caches than the root cache.

Fig. 4.5(a) shows that, when $\mu_e = 0$, the SSAHD for LCD and LCE become closer as $C_R$ increases. This is because as $C_R$ increases, the SSHR-R for LCE increases rapidly compared to the SSHR-R for LCD.

Fig. 4.5(a) shows that, when $\mu_e = 0$, PCO achieves the best SSAHD, especially for large $C_R$. Compared to LCE, PCO has the same SSHR-L and a better SSHR-R. This
is due to the fact that PCO allows any requested object to be stored in the requesting leaf cache (like LCE), while preventing objects that are effectively one-timers from being stored in the root cache, as discussed in Section 4.3.1.

Compared to LCD, PCO achieves a lower SSHR-L and a much higher SSHR-R. On the other hand, Fig. 4.5(a) shows that a user under PCO experiences a smaller average hit distance, especially when \( C_R \) is large.

Fig. 4.5(b) shows that as \( \mu_e \) increases, the SSAHD for PCO degrades rapidly, especially for large \( C_R \). In this case, the objects expire in the leaf caches before getting promoted to the root cache. Thus, fewer objects are cached by the root cache as \( \mu_e \) increases. As \( C_R \) increases, PCO does not improve since only a small portion of popular objects, with a request rate high enough to exceed the expiry rate, can be cached in the root cache simultaneously. Hence, PCO does not make use of the extra available cache capacity. Similarly, Fig. 4.5(b) shows that as \( \mu_e \) increases, the SSAHD for LCD degrades rapidly because fewer objects can be simultaneously cached in the leaf caches. Fig. 4.5(b) shows that, when \( \mu_e = 1 \), LCE achieves better SSAHD than LCD. This is because the SSHR-L for LCE is better than SSHR-L for LCD, while both LCE and LCD have similar SSHR-R.

4.5.2 Evaluation of the SSAHD under TTL-T

Fig. 4.6 shows that the analysis accurately estimates the SSAHD for LCE under TTL-T, as a function of \( \mu_e, C_R, \) and \( \eta \).

Fig. 4.6 shows that, under TTL-T, PCO outperforms both LCE and LCD at \( \mu_e = 1 \), even for high \( \eta = 0.8 \). Comparing Fig. 4.5(b) and Fig. 4.6, the SSHR-R of
Figure 4.6: SSHR (in percent) and SSAHD or LCE, LCD, and PCO under TTL-T, where $\mu_e = 1$.

PCO under TTL-T improves greatly, especially for large $C_R$. This is because, unlike TTL-IE, TTL-T allows the leaf caches to store the objects for a longer time, even after they expire. This allows more objects to get promoted by the leaf caches. Also, as $C_R$ decreases, the difference between the SSHR-R of PCO under TTL-IE and TTL-T diminishes, since the root cache keeps fewer expired objects.

Also, comparing Fig. 4.5(b) and Fig. 4.6 shows that the leaf cache performance for LCE and PCO varies according to the TTL implementation. For example, the
SSHR-L for LCE and PCO under TTL-IE is 12.49%. Under TTL-T, the SSHR-L increases to 12.71% when \( \eta = 0.2 \), while it decreases to 11.29% when \( \eta = 0.8 \). This is consistent with SCS results discussed in Section 3.9.

Fig. 4.6(a) shows that, unlike TTL-IE in Fig. 4.5(b), the SSAHD for LCD is similar to the one for LCE even for small \( C_R \) when \( \eta = 0.8 \). Furthermore, Fig. 4.6(b) shows that as \( \eta \) decreases to 0.2, LCD outperforms LCE, especially for small \( C_R \). This is because, as \( \eta \) decreases, it becomes more likely that the expired objects in the leaf caches are valid, which improves the SSHR-L for LCD.

4.5.3 Evaluation of the IAHD under NEM

In this section, the IAHD of the HCS is evaluated under a non-stationary access pattern. The scenario in Fig. 1.10(b), where the HCS operating in the steady state enters a transient period, is considered.

In this section, it is assumed that the number of objects in the time interval \((0, t_v)\) is 5000. The number of objects at the OS increases by \( \Delta M \) at time \( t_v = 30 \) hours, which is long enough to allow the cache hierarchy to reach steady state. It is assumed that the \( \Delta M \) newly generated objects become the new most popular objects. Therefore, the popularity of the objects in the cache at \( t_v \) degrades rapidly after \( t_v \).

Figs. 4.7 and 4.8 show that the analysis accurately estimates the IAHD and the IHR for LCE and PCO for various values of \( \Delta M \) and \( C_R \).

Fig. 4.7(a) shows that, when \( \Delta M = 50 \) and \( C_R = C_L = 50 \), LCE is very robust and the generation of the new objects has a little effect on the IAHD. Moreover, PCO is more robust than LCD in the sense that PCO restores its initial SSAHD very
Figure 4.7: IHR (in percent) and IAHD of LCE, LCD, and PCO, where $C_R = 50$ and $\mu_e = 0$.

quickly. Fig. 4.7(a) shows that PCO restores its superiority over LCE and LCD in a very short time (at time 30.5 hours), while LCD takes longer to return to its SSAHD. This is because the IHR-L for PCO is greater than the IHR-R for LCD. Hence, the new popular objects will be promoted to the PCO root cache faster than moving the objects down to the LCD leaf caches.

Fig. 4.7(b) shows that, when $C_R = C_L = 50$ and $\Delta M = 250$, the IAHD for both PCO and LCD decreases sharply after the generation of the new objects at $t_v = 30$
Figure 4.8: IHR (in percent) and IAHD of LCE, LCD, and PCO, where $C_R = 250$ and $\mu_e = 0$.

hours. However, like the results in Fig. 4.7(a), Fig. 4.7(b) shows that PCO adapts much faster to this change in the access pattern compared to LCD, which takes longer to outperform LCE.

Fig. 4.8(a) shows that, when $\Delta M = 50$ and $C_R$ is 250, PCO experiences a sharp increase in IAHD after $t_v = 30$ hours. However, like Fig. 4.7(a), Fig. 4.8(a) shows that PCO adapts very quickly and achieves the best IAHD starting from 30.5 hours.

In Fig. 4.8(b), it is observed that the robustness of PCO degrades rapidly compared
to LCE and LCD. In this case, PCO takes an hour to outperform LCE and LCD.

Unlike Fig. 4.7, Fig. 4.8 shows that LCD restores its initial IAHD faster than PCO when $C_R = 250$. In this case, the LCD root cache has more room to keep the new popular objects generated at $t_v$ for a longer time. This results in a sharp increase in IHR-R for LCD, and allows the LCD root cache to move more new popular objects to the leaf caches. On the other hand, PCO takes longer to fill the large root cache with the 250 new popular objects, which have to be promoted through smaller leaf caches.

4.5.4 Impact of TLCs under NEM and TTL-IE

Similar to Section 4.5, simulations have been used to verify the results of the proposed analysis for LCE and PCO and to compare them with LCD under TTL-IE, assuming TLCs. The analysis and simulator model the TLCs population, assuming that $\lambda_u = 10$ and $\mu_u = 11$. Thus, when the population reaches steady state, the average number of TLCs in the HCS is $\frac{\lambda_u}{\mu_u - \lambda_u} = 10$. In steady state, it is assumed that the total request rate is $\beta = 2000$ requests/hour. Note that it is assumed that the request rate $\beta$ is fixed regardless of the number of available TLCs.

Fig. 4.9 shows that the analysis accurately estimates the SSAHD for LCE and PCO as a function of $\mu_e$ and $C_R$, assuming TLCs.

Fig. 4.9(a) shows that, at $\mu_e = 0$, PCO achieves much better SSAHD, compared to LCE and LCD. In PCO, promoting the popular objects to the fixed root cache allows them to be available to the users in the HCS for a longer time. On the other hand, the SSAHD for LCD degrades dramatically, since LCD depends on storing
Figure 4.9: SSHR (in percent) of LCE, LCD, and PCO under TTL-IE and with TLCs.

these popular objects in the TLCs. Thus, LCD loses these popular objects as the TLCs leave the network. Furthermore, the LCD root cache can easily get polluted by objects that are effectively one-timers, which limits the backup role that the root cache plays for PCO. Note that Fig. 4.9(a) shows that LCE is less sensitive than LCD to the TLCs disconnections, and thus, unlike FCs, LCE achieves similar SSAHD as LCD in the TLCs context.

Also, similar to Fig. 4.5(b), Fig. 4.9(b) shows that as $\mu_e$ increases, the SSAHD for
PCO degrades rapidly under TTL-IE, especially for large $C_R$.

4.6 Summary

In this chapter, a new analytical model for a two-level HCS using LCE was proposed. Moreover, the design and analysis of a new HCS sharing protocol, PCO, were presented. In Section 4.5, the considered sharing protocols were evaluated under NEM, TTL-IE, and TTL-T consistency. Moreover, the impact of TLCs was studied.

The results in this chapter showed that the SSAHD of the considered sharing protocols depends on the object expiry rate and the TTL implementation. Assuming NEM, the results showed that PCO achieves the best SSAHD followed by LCD then LCE. This is especially true when the root cache is large. The results showed that this is also true under TTL-T. However, under TTL-IE, the SSAHDs of PCO and LCD degrade rapidly as the object expiry rate increases. Furthermore, assuming NEM, the results showed that PCO also improves the SSAHD in the TLCs context.

Moreover, the considered sharing protocols were evaluated under a non-stationary access pattern. Assuming NEM, the results showed that PCO is more robust than LCD when new popular objects are generated, especially when the root cache is small. When the root cache capacity is large, as the number of the newly generated object increases, LCD and LCE become more robust than PCO at the expense of the SSAHD.
Chapter 5

Analysis of the Distributed Cache System (DCS)

This chapter presents an analytical model for estimating the instantaneous hit ratio (IHR) and the instantaneous average hit distance (IAHD) of the proposed distributed caching system (DCS), assuming fixed caches (FCs) or temporary caches (TCs). The analytical model in this chapter is based on the analysis introduced for the SCS and the HCS in Chapter 3 and Chapter 4, respectively.

Fig. 1.4 shows the proposed DCS with FCs where each group of users (i.e. in a community, or an institution) is served by a fixed cache (local cache), while local caches cooperate to improve the user’s perceived latency. As discussed in Section 1.1.3, a user may only query its local cache (the closest cache in the DCS), while the local cache queries other caches (remote caches) in the DCS, if it does not have the requested object.

Moreover, the DCS with temporary caches (TCs) is considered, as discussed in Section 1.1.4. In this case, caching activity within a single MANET, which consists of a group of mobile users (e.g. laptops) who may host TCs, is modeled. Similar to the proposed analysis for the HCS with TLCs in Section 4.4, the birth-death connectivity model is assumed to model the TCs populations in the DCS, as will be discussed in detail in Section 5.2.

As discussed in Section 1.1.3, the proposed DCS (with FCs or TCs) adopts the same rules specified by ICP, where multiple copies of the same object may be stored.
at different locations within the DCS. Also, it is assumed that LRU is applied independently at each cache\(^1\), such that: (1) a cache does not cache an object based on a remote request; (2) the local LRU list is not affected by remote hits; and (3) under TTL-T, the cache does not validate an expired object due to a remote request.

Moreover, a modified version of ICP, where the object that is served by a remote cache is not locally cached, is evaluated using simulations. In this case, a cache in the DCS stores only the objects that are served by the OS. This is called ICP-cache origin copy only (ICP-COCO). Under TTL-IE, ICP-COCO implies that only one copy of the same object may exist in the DCS at a time (i.e. follows the 1CH), and thus, the available cache capacities in the DCS are used more efficiently. On the other hand, this is not true for TTL-T, where multiple copies of the same object may exist in different locations in the DCS using ICP-COCO.

The remainder of this chapter is organized as follows. Sections 5.1 and 5.2 present the analysis for the proposed DCS assuming FCs and TCs, respectively. Then, Section 5.3 discusses the evaluation results for the proposed DCSs. Finally, the results are summarized in Section 5.4.

5.1 Analysis of Fixed Caches (FCs)

This section introduces an analytical model for estimating the IAHD of the DCS using ICP, as shown in Fig. 1.4. The analysis in this chapter adopts the access pattern characteristics introduced for the HCS in Section 4.1. Thus, the analysis in

\(^1\)The analysis introduced in this chapter is also applicable for Infinite Cache and FIFO. However, this chapter merely considers LRU for conciseness.
this chapter assumes homogenous populations are connected to $K$ fixed caches (FCs) that have the same capacity, $C$, and are always connected to the DCS. Hence, all caches experience the same request rate, $\beta_L = \beta/K$, with the same Zipf slope, $\alpha$, such that the local average request rate of object $i$ is

$$\lambda_L(i) = \frac{\beta_L}{\sigma i^\alpha} \quad (5.1)$$

The local IHR, $H_L(t)$, can be calculated as

$$H_L(t) = \sum_{i=1}^{M} \frac{1 - S_L(t, i)}{\sigma i^\alpha} \quad (5.2)$$

where $S_L(t, i)$ is the probability that object $i$ is not served by the local cache at time $t$, which can be calculated under NEM, TTL-IE, or TTL-T using the analysis proposed for the SCS in Section 3.5.

Furthermore, under NEM and TTL-IE, the probability that the request from a cache $k$ for object $i$ at time $t$ is not served by any remote cache, $q \neq k$, in the DCS, $S_R(t, i)$, can be calculated as

$$S_R(t, i) = \prod_{q=1, q \neq k}^{K} S_q(t, i) = \prod_{q=1, q \neq k}^{K} P_q(t, i, 0) \quad (5.3)$$

Assuming homogenous caches experience the same access patterns, such that $S_q(t, i) = S_L(t, i) \forall q$, $S_R(t, i)$ is

$$S_R(t, i) = S_L(t, i)^{K-1} = P_L(t, i, 0)^{K-1} \quad (5.4)$$

Then, the remote IHR of the cache $k$, $H_R(t)$, will be calculated using $S_R(t, i)$, such
Under TTL-T, the same analysis of TTL-IE is applied. However, $S_R(t, i)$ is calculated after modifying (5.4), such that

$$S_R(t, i) = \prod_{q=1, q \neq k}^K (P_q(t, i, 0) + \sum_{j=1}^C P_q(t, i, 2j))$$

(5.6)

Assuming homogenous caches experience the same access patterns, $P_q(t, i, j) = P_L(t, i, j) \forall q$. Hence,

$$S_R(t, i) = (P_L(t, i, 0) + \sum_{j=1}^C P_L(t, i, 2j))^{K-1}$$

(5.7)

where a remote cache $q$ serves the requested object, if and only if the cached object is fresh. If the cached object is expired, the remote cache will not validate the expired object. Note also that, according to the ICP, the local cache queries other caches in the DCS, if and only if the requested object is not in the local cache. Hence, (5.5) is also used to calculate the $H_R(t)$ under TTL-T.

5.2 Analysis of Temporary Caches (TCs)

This section introduces an analytical model for estimating the IHR of the DCS using ICP, assuming TCs. As discussed in Section 1.1.4, the proposed network model considers a group of mobile users (e.g. vehicles, laptops, etc.) that form a MANET, which maintains the communication between the mobile users and also provides the
users with Internet connections. As illustrated in the VANET example in Fig. 1.7, the proposed DCS assumes that there are some mobile users that have extra power and cache storage to host TCs.

In the proposed DCS, the TCs cooperate with each other to serve mobile users in the MANET, which improves the user’s perceived latency, reduces the load on the infrastructure, and improves the network capacity. This is particularly useful in periods of high load where a dense population is interested in accessing the same objects. For example, during a sporting event or a concert where users are accessing the same highlight footage at slightly different times, a small number of objects are heavily requested by many users over a short time-span. In this scenario, the proposed DCS prevents users from overwhelming the limited infrastructure bandwidth and allows more users to simultaneously access the requested objects, as shown in Fig. 5.1.

The DCS with TCs follows the same network model and assumptions described in Section 4.4.1. However, unlike the HCS, the proposed DCS consists of multiple TCs in a single MANET. Fig. 5.2 shows the proposed DCS where each TC is associated with a group of users who query that TC (local TC), rather than the access point. Like the HCS, when a TC disconnects from the DCS, its users will be automatically associated with the closest available TC in the MANET. When a new TC joins the DCS, nearby users will automatically be associated with this new TC.

As discussed in Section 1.1.4, the user’s request is satisfied from the local TC, if the TC has the requested object. Otherwise, the local TC will query the other TCs (remote TCs) in the DCS for that object. If the object cannot be found in any TC (remote TC) in the DCS, the TC downloads the requested object from the OS.
via an infrastructure access point and forwards the object to the requesting user (or the access point just forwards the object to the user and its TC). It is important to note that the DCS adopts ICP where it is assumed that the remote TC serves a user through the user’s local TC, which will store the requested object for future requests. This constraint is relaxed in ICP-COCO, where the local TC does not cache objects that are served by the remote TCs. Hence, the TC may only cache those objects that are served by the OS, as shown in Fig 5.3. In Section 5.3, the steady state performances of ICP and ICP-COCO will be evaluated.

For DCS using ICP, in order to estimate the local IHR of a TC, $H_L(t)$, the

Figure 5.1: Network topology for temporary caches in the DCS.
Figure 5.2: The proposed distributed TCs network model.

analysis introduced in Section 4.4 for the HCS with temporary leaf caches (TLCs) can be used, under TTL-IE or TTL-T. Note that, similar to the analysis introduced in Section 4.4, the TCs population is modeled as a birth-death process, with an arrival rate that equals $\lambda_u$ and a departure rate that equals $\mu_u$.

However, in order to calculate the DCS remote IHR, $H_R(t)$, the analysis has to account for the fact that the remote hit occurs when any remote TC in the DCS has the requested object. Thus, the remote IHR, $H_R(t)$, is calculated using the analysis.
introduced in Section 5.1 for the DCS with FCs. Hence, under NEM and TTL-IE, $S_R(t, i)$ is calculated using (5.4), whereas, under TTL-T, $S_R(t, i)$ is calculated using (5.7), assuming homogenous TCs that experience the same access patterns.

5.3 Steady State Evaluation of the Proposed DCS

Similar to the evaluation for the HCS in Section 4.5, the workload generator is used to generate 30 workload profiles that statistically conform with the mathematical models adopted in this chapter. Each workload profile consists of a sequence of requests. Each request in the generated workload includes requested object id, object expiry time, requesting TC id, and the identities of the TCs that are currently connected to the DCS. The simulator is used to calculate the local and remote SSHR for each cache in the DCS. The final simulated SSHR is then determined by averaging the SSHR for all the simulated profiles. The final simulated SSAHD is calculated using the simulated SSHR values in (4.1). Note that only the steady state performance of the DCS is evaluated in this section, while the robustness of the proposed DCS under a non-stationary access pattern will be considered in future work.
For FCs and TCs, it is assumed that each cache has the same capacity, $C$, and the same hop distances to the OS, $D_O = 10$. The hop distance between any two caches in the DCS is also constant and is equal $D_R = 2$, and the hop distance to the local cache is $D_L = 0$.

The SSAHD is evaluated assuming that all the objects have the same expiry rate $\mu_e$ (i.e. $\mu_e = \mu(i) \forall i$). Analysis and simulation factors and their levels are summarized in Table 5.1. Note that the levels presented in Table 5.1 only aim at illustrating the concepts introduced in this chapter and do not mimic any realistic workload.

**Table 5.1: DCS: evaluation factors and levels.**

<table>
<thead>
<tr>
<th>Factor</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of objects</td>
<td>$M$</td>
<td>1000</td>
</tr>
<tr>
<td>Number of Caches</td>
<td>$K$</td>
<td>10</td>
</tr>
<tr>
<td>Zipf Slope</td>
<td>$\alpha$</td>
<td>0.8 [14]</td>
</tr>
<tr>
<td>Request Rate</td>
<td>$\beta$</td>
<td>1000 request/hr</td>
</tr>
<tr>
<td>Cache Capacity</td>
<td>$C$</td>
<td>10-50 Objects</td>
</tr>
<tr>
<td>Object Expire Rate</td>
<td>$\mu_e$</td>
<td>1, 10 object/hr</td>
</tr>
</tbody>
</table>

Also, for the DCS with TCs, this section follows the TCs population model in Section 4.5.4, where $\lambda_u = 10$ and $\mu_u = 11$ and the total request rate in steady state is $\beta = 1000$ requests/hour.

In the following, Section 5.3.1 presents the evaluation results for the DCS using ICP, assuming FCs or TCs. Then, Section 5.3.2 discusses the evaluation results for the DCS using ICP-COCO, assuming FCs or TCs. In all the plots in the following sections, the data points are generated using simulation and the solid lines are generated using analysis.
5.3.1 Evaluation of ICP

Figs. 5.4, 5.5, and 5.6 show the excellent match between analytical and simulation results of ICP at different values of $C$, $\mu_e$, $\eta$ for FCs or TCs.

Figure 5.4: SSHR (in percent) and SSAHD of the DCS with FCs, using ICP under TTL-IE or TTL-T, where $\mu_e = 1$.

Figure 5.5: SSHR (in percent) and SSAHD of the DCS with TCs, using ICP under TTL-IE, where $\mu_e = 1$.

Fig. 5.4 shows that when $\mu_e = 1$, TTL-IE outperforms TTL-T in terms of the SSAHD, assuming FCs. Fig. 5.4 shows that when $\eta = 0.2$, TTL-IE and TTL-T have
Figure 5.6: SSHR (in percent) and SSAHD of the DCS with FCs, using ICP under TTL-IE or TTL-T, where $\mu_e = 10$.

similar local SSHR. Under TTL-T, the SSHR of a local cache decreases as $\eta$ increases. This is consistent with the result for the SCS in Section 3.9. Also, Fig. 5.4 shows that TTL-IE yields a better remote SSHR than TTL-T. This is because ICP does not allow remote requests to validate the expired objects. Hence, since the remote hit occurs only if the requested object is fresh, TTL-IE that allows only fresh object to linger in remote caches yields more remote hits. These results for FCs are also true for TCs, as shown in Fig. 5.5. However, for TCs, the SSAHD increases as a result of TCs disconnections.

Fig. 5.6 shows that when $\mu_e = 10$, assuming FCs, TTL-T outperforms TTL-IE in terms of SSAHD if $\eta$ is small (e.g. 0.2) and $C$ is large (e.g. 50). Fig. 5.6 shows that TTL-T achieves a better local SSHR than TTL-IE, regardless of the value of $\eta$. The superiority of TTL-T in terms of local SSHR increases as $C$ increases. Again, this is consistent with the results in Section 3.9. However, Fig. 5.6 shows that TTL-IE achieves a better remote SSHR than TTL-T. Since the objects in remote caches are
more likely to be expired as $\mu_e$ increases, TTL-T allows remote caches to be polluted (from the local cache perspective) by more expired objects that cannot be validated.

Moreover, the remote SSHR under TTL-T greatly degrades as $C$ increases. This is because as $C$ increases, the local cache keeps more expired objects that are validated directly from the OS, which reduces the request rate for the remote caches. Moreover, as $C$ increases, the local cache hits increase, which also reduces the request rate for the remote caches. Fig. 5.6 shows that when $\eta = 0.8$, as $C$ increases, the local SSHR increases slowly, while the remote SSHR decreases quickly. This results in an interesting behavior for TTL-T with $\eta = 0.8$, where the lowest SSAHD of 7.7 is achieved when $C = 20$, while the SSAHD increases as $C$ increases (e.g. the SSAHD is 8 when $C = 50$), as illustrated in Fig. 5.6.

5.3.2 Evaluation of ICP-COCO

Fig. 5.7 shows that when $\mu_e = 1$, ICP-COCO under TTL-IE outperforms ICP-COCO under TTL-T in terms of SSAHD, assuming FCs. However, unlike ICP, ICP-COCO under TTL-T achieves a better local SSHR than ICP-COCO under TTL-IE. Since the cache under TTL-T does not validate an expired object due to a remote request, ICP-COCO under TTL-T allows multiple copies of the same object to be cached at different locations in the DCS. This redundancy plays a key role in improving the local SSHR, since it increases the average lifetime of an object within the DCS. On the other hand, TTL-IE degrades the local SSHR, since it allows only one copy to be cached at a time. To better explain this point, let $N(i)$ denote the number of copies of object $i$ that are currently cached. Under TTL-T, the average lifetime of a cached
object $i$ in the DCS, $L(i)$, is a random variable that is equivalent to the maximum of $N(i)$ exponential random variables that have the same rate $\mu_e$. In this case, $L(i)$ can be calculated using (5.8) [153].

$$L(i) = \sum_{x=1}^{N(i)} 1/x\mu_e$$  \hspace{1cm} (5.8)

According to (5.8), for TTL-IE (i.e. $N(i) \leq 1$), the average lifetime of $1/\mu_e$ is obtained. Thus, TTL-T outperforms TTL-IE, since $L(i)$ is proportional to the number of object copies, $N(i)$.

Figure 5.7: SSHR (in percent) and SSAHD of the DCS with FCs, using ICP-COCO under TTL-IE or TTL-T, where $\mu_e = 1$.

Furthermore, Fig. 5.7 shows that, like ICP, ICP-COCO under TTL-IE achieves a better remote SSHR than ICP-COCO under TTL-T, when $\mu_e = 1$. The first reason is that ICP-COCO under TTL-IE allows only a single copy of each object to be cached at a time. Hence, TTL-IE uses the available cache capacities on the remote caches more effectively than TTL-T. Second, ICP-COCO under TTL-T wastes the cache capacity of remote caches by allowing expired objects to be stored, as discussed in
Section 5.3.2.

The results for FCs in Fig. 5.7 are also true for TCs, as shown in Fig. 5.8 where TTL-IE achieves a better remote SSHR and a better SSAHD, while TTL-T achieves a better local SSHR. However, for TCs, the local SSHR of TTL-T becomes closer to the SSHR of TTL-IE, since the advantage of allowing multiple copies under TTL-T vanishes due to TCs disconnections. Also, for TCs, TTL-IE and TTL-T achieve similar remote SSHRs. This is because under TTL-IE, the DCS loses single copies of popular objects quickly due to TCs disconnections, while allowing multiple copies under TTL-T provides a backup for the loss of popular objects.

Figure 5.8: SSHR (in percent) and SSAHD of the DCS with TCs, using ICP-COCO under TTL-IE, where $\mu_e = 1$.

Fig. 5.9 shows that when $\mu_e = 10$, ICP-COCO under TTL-T outperforms ICP-COCO under TTL-IE in terms of SSAHD if $\eta$ is small (e.g. 0.2) and $C$ is large, assuming FCs. This is similar to the results obtained for ICP in Fig. 5.6. However, note that the SSAHD of TTL-T when $\eta = 0.8$ becomes closer to the SSAHD of TTL-IE when ICP-COCO is used. This is because of the rapid increase in the local SSHR
of ICP-COCO under TTL-T. Furthermore, like ICP, TTL-T with $\eta = 0.8$ achieves the lowest SSAHD of 7.85 at $C = 30$. Then, the SSAHD increases as $C$ increases (e.g. at $C = 50$, the SSAHD is 8).

![Graph showing SSHR and SSAHD](image)

Figure 5.9: SSHR (in percent) and SSAHD of the DCS with FCs, using ICP-COCO under TTL-IE or TTL-T, where $\mu_e = 10$.

Note that, compared to ICP, ICP-COCO improves the SSAHD of the DCS with FCs, if $\mu_e = 0$ (i.e. under NEM). Under NEM, ICP-COCO uses the cache capacity more efficiently by allowing at most one copy of an object to be stored in the DCS. However, this is not true under TTL-IE, since the multiple copies allowed by ICP increase the average object lifetime in the network, according to (5.8). Table 5.2 illustrates the SSAHD of ICP-COCO and ICP under NEM and TTL-IE, when $\mu_e = 1$. Furthermore, the multiple copies provide backup for the loss of the popular objects in case of DCS with TCs. Hence, ICP achieves a better SSAHD than ICP-COCO under NEM as well as TTL-IE, as shown in Table 5.3.
Table 5.2: SSAHD of ICP and ICP-COCO under NEM and TTL-IE, where $\mu_e = 1$, assuming FCs.

<table>
<thead>
<tr>
<th>$C$</th>
<th>NEM ICP</th>
<th>NEM ICP-COCO</th>
<th>TTL-IE ICP</th>
<th>TTL-IE ICP-COCO</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>5.37</td>
<td>5.2</td>
<td>5.42</td>
<td>5.41</td>
</tr>
<tr>
<td>40</td>
<td>4.84</td>
<td>4.58</td>
<td>4.91</td>
<td>4.98</td>
</tr>
<tr>
<td>50</td>
<td>4.40</td>
<td>3.98</td>
<td>4.48</td>
<td>4.86</td>
</tr>
</tbody>
</table>

Table 5.3: SSAHD of ICP and ICP-COCO under NEM and TTL-IE, where $\mu_e = 1$, assuming TCs.

<table>
<thead>
<tr>
<th>$C$</th>
<th>NEM ICP</th>
<th>NEM ICP-COCO</th>
<th>TTL-IE ICP</th>
<th>TTL-IE ICP-COCO</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>6.42</td>
<td>6.58</td>
<td>6.48</td>
<td>6.79</td>
</tr>
<tr>
<td>40</td>
<td>6.06</td>
<td>6.33</td>
<td>6.18</td>
<td>6.65</td>
</tr>
<tr>
<td>50</td>
<td>5.79</td>
<td>6.15</td>
<td>6.0</td>
<td>6.49</td>
</tr>
</tbody>
</table>

5.4 Summary

In this chapter, the steady state performance of the DCS using ICP and ICP-COCO was evaluated. The results in this section were as follows:

- For both ICP and ICP-COCO, TTL-IE yields a better SSAHD than TTL-T, when $\mu_e$ is moderate (e.g. 1 per hour). This is true for FCs as well as TCs. However, for FCs, TTL-T yields a better SSAHD than TTL-IE when $\mu_e$ is high (e.g. 10 per hour) and the validation factor, $\eta$, is small.

- For both ICP and ICP-COCO, TTL-IE yields a better remote SSHR than TTL-T. This is because a remote hit only occurs if the requested object in the remote cache is fresh. Hence, with respect to a local cache, TTL-T allows remote caches to be polluted by expired objects that cannot be validated. This greatly degrades the
remote SSHR under TTL-T, especially when $\mu_e$ is high.

- For both ICP and ICP-COCO under TTL-T, when $\eta = 0.8$ and $\mu_e = 10$, as $C$ increases, the local SSHR increases, while the remote SSHR decreases rapidly. This results in an interesting behavior where the SSAHD increases as a function of $C$.

- For ICP-COCO, TTL-T improves the local SSHR assuming FCs or TCs. This is true especially for FCs when $\mu_e$ is high (e.g. 10 per hour), which allows TTL-T to achieve a better SSAHD than TTL-IE when $\eta$ is small.

- For FCs under NEM, ICP-COCO outperforms ICP since ICP-COCO utilizes the cache capacities more efficiently by allowing only one copy of each object to be cached at a time. However, for TCs, or under TTL-IE, ICP outperforms ICP-COCO since allowing multiple copies plays a crucial role either to backup the loss of popular objects due to disconnections, in case of TCs, or due to expiry, in case of TTL-IE.
Conclusions and Future Work

In this chapter, the conclusions of this dissertation are presented in Section 6.1. Then, the future work is discussed in Section 6.2.

6.1 Conclusions

Caching Web objects close to end users allows users’ requests to be satisfied from a nearby Web cache rather than a faraway origin server (OS). Cache hits improve the user’s perceived latency, reduce the load on the OS, and achieve bandwidth savings over costly Internet links. In this dissertation, three Web cache systems are considered: a single cache system (SCS), a hierarchical cache system (HCS), and a distributed cache system (DCS).

In Chapter 3, an analytical model for estimating the instantaneous hit ratio (IHR) of the SCS was introduced. Estimating the IHR is important for any application where the number of objects, object popularity, and/or the caching resources themselves are subject to change. Consider a scenario where objects experience only short-term popularity, such as a group of users watching sports highlights or accessing vehicular traffic information. Here, it is important for a cache to reach maximum performance quickly before the popularity of the cached objects starts to drop. Moreover, a vehicular or mobile network may store cached objects on mobile devices. In this case, it is also important for the cache to reach a high hit ratio quickly before the mobile
device that is hosting the cache disconnects from the network.

Chapter 3 was focused on evaluating how the cache hit ratio evolves as a function of time in two cases. The first case assumed a stationary access pattern where the number of objects and the object popularity are fixed throughout the evaluation interval. In this case, the IHR was evaluated in the transient period starting from an empty cache. The second case adopted more realistic assumptions by assuming that the number of objects increases periodically (i.e. non-stationary access pattern). It was assumed that the most recently generated objects become the most popular ones (news headlines, for example). Hence, the popularity of the formerly cached objects decreases every time new objects are generated. In this case, the cache may not reach the steady state and the only way to quantify the hit ratio experienced by users is to calculate the IHR.

Moreover, the SCS analysis in Chapter 3 accommodates either an infinite or a finite cache capacity. For the finite case, the cache applies a replacement policy to decide which objects to keep. The proposed analysis introduced a solution for two replacement policies: LRU and FIFO. Furthermore, the proposed analysis considered a typical TTL mechanism (i.e. TTL-T), whereby the request for an expired cached object results in sending a HTTP conditional GET request to the OS. In this case, the replacement policy does not distinguish expired objects from the fresh ones. This was compared to a second TTL implementation where cached objects are ejected immediately once they expire (i.e. TTL-IE). In this second case, the replacement policy is applied only to fresh objects, since expired objects do not exist in the cache. Also, the case where objects do not expire is considered (i.e. NEM).
Furthermore, Chapter 3 introduced a new replacement policy, called frequency-based-FIFO (FB-FIFO). FB-FIFO improves the IHR of FIFO by creating a variable-sized protected cache segment for objects that are not effectively one-timers. The results showed that the replacement policy using the two implementations for TTL (i.e. TTL-T and TTL-IE) yields different IHRs, especially for large caches. Under TTL-IE or TTL-T, the results showed that FB-FIFO outperforms the steady state hit ratio (SSHR) of FIFO and LRU.

The results in Chapter 3 illustrated that the considered replacement policies have very different transient performance, whether the popularity of cached objects is fixed or not. This makes the proposed analysis valuable when considering scenarios like changing popularity or mobile caching. Under TTL-IE, assuming a stationary access pattern, the results showed that the hit ratio reaches a steady state within a period that depends on cache capacity, object expiry rate, and request rate. In this case, the results showed that FB-FIFO outperforms FIFO and LRU in the steady state, especially for small cache capacities. On the other hand, these results change when the OS periodically generates new popular objects. Furthermore, the results showed that when the OS generates new popular objects periodically, the users may experience different hit ratios when they initiate requests at different times. In this case, the hit ratio experienced by these users can only be quantified by calculating the IHR.

In Chapter 4, an analytical model for estimating the instantaneous average hit distance (IAHD) of a two-level HCS was proposed. The proposed analytical model considers LCE sharing protocol under NEM, TTL-IE, and TTL-T. Also, the design and analysis of a new hierarchical sharing protocol, called PCO, were introduced.
Simulations were used to validate the analytical results for LCE and PCO and to compare them with LCD.

The results in Chapter 4 showed that the steady state performance of the considered sharing protocols depends on the object expiry rate and the adopted TTL implementation. Assuming NEM, the results showed that PCO achieves the best steady state average hit distance (SSAHD) followed by LCD then LCE. This is especially true when the root cache is large. The results showed that this is also true under TTL-T. However, under TTL-IE, the SSAHDs of PCO and LCD degrade rapidly as the object expiry rate increases. Also, under NEM, the results showed the superiority of PCO for the HCS with temporary leaf caches (TLCs) that suffer from frequent disconnects.

Moreover, in Chapter 4, the robustness of the considered sharing protocols was evaluated assuming a non-stationary access pattern. Assuming NEM, the results showed that PCO is more robust than LCD when new popular objects are generated, especially when the root cache is small. However, when the root cache capacity is large, and as the number of newly generated objects increases, LCD and LCE become more robust than PCO, at the expense of the SSAHD.

In Chapter 5, an analytical model for estimating the IAHD of a DCS using ICP was proposed. The proposed analysis for the DCS considered LRU caches under NEM, TTL-IE, and TTL-T. Moreover, a modified version of the ICP, called ICP-COCO, where the object that is served by a remote cache is not locally cached, was evaluated using simulations. ICP-COCO under NEM or TTL-IE allows only one copy of the same object to exist in the DCS at a time, while this is not true for ICP-COCO
under TTL-T.

The results in Chapter 5 showed that, for both ICP and ICP-COCO, TTL-IE yields a better SSAHD than TTL-T, when the object expiry rate, $\mu_e$, is moderate (e.g. 1 per hour). This is true for a DCS with either fixed caches (FCs), or temporary caches (TCs). However, for FCs, TTL-T yields a better SSAHD than TTL-IE when $\mu_e$ is high (e.g. 10 per hour) and the validation factor, $\eta$, is small. Furthermore, the results for FCs showed an interesting behavior where the SSAHD increases as the cache capacity increases. This occurs for both ICP and ICP-COCO under TTL-T, when the validation factor, $\eta = 0.8$, and the object expiry rate, $\mu_e = 10$. Moreover, the results also showed that, for FCs under NEM, ICP-COCO outperforms ICP. However, this is not true for TCs, or under TTL-IE.

6.2 Future work

The findings in this dissertation are valid under the independent reference model (IRM), which is widely used to develop tractable analytical cache models. However, the IRM might not accurately model empirical traces that have varying degrees of temporal correlation between object requests, as discussed in Section 2.3. Thus, an interesting extension of the proposed model would be to incorporate short term popularity [14,62,109]. For example, the LRU stack in [14] maintains an ordered list of documents based on recency of access, which can be modeled using the proposed LRU Markov chain. In a static LRU stack, the probability of referencing is associated with the stack position, whereas in a dynamic LRU stack, the probability of referencing
depends on the object identity as well as the stack position [106].

Furthermore, in the proposed LRU Markov chain, the probability that an object is requested depends entirely on the object identity. Therefore, the main difference between the LRU stack models and the proposed LRU Markov chain is that in LRU stack, the request rate of object $i$, $\lambda(i)$, is a function of the current state $j$. For example, let $\lambda(i, j)$ denote the request rate of object $i$ that is currently occupying state $j$ in the LRU stack. Then, $\lambda(i, j)$ can be calculated as $\lambda(i, j) = \beta f(i, j)$, where $f(i, j)$ is the probability that object $i$ that is currently in state $j$ will be requested. Note that in the proposed LRU Markov chain $f(i, j) = 1/(\sigma i^a)$, which is independent of $j$, as discussed in Section 3.1.

For the HCS, four main issues will be considered in future work. First, it is important to evaluate the considered sharing protocols using more than two cache levels. This can be done based on the concepts proposed for two-level hierarchy in Chapter 4. Second, individual caches at different levels may follow different consistency models and it would be interesting to evaluate LCE, PCO, and LCD in this case. Also, evaluating the HCS, with each level running a different replacement policy can be done using the proposed analysis in Chapter 4. Third, the proposed analysis in Chapter 4 for PCO under TTL-IE can also be extended for PCO under TTL-T. Finally, incorporating more realistic connectivity model for TLCs into the proposed simulation and analysis remains an interesting point. It will be interesting to evaluate the considered sharing protocols using more realistic mobility models, such as Random Waypoint and Random Walk [154,155,156]. Also, comparing the analytical results to trace-based simulations, which use real traces, will be considered.
For the DCS, the steady state evaluation was presented in Chapter 5, while evaluating the robustness of the proposed DCS under a non-stationary access pattern will be considered in future work. Also, comparing the IAHDs of the proposed HCS and DCS will be considered. Moreover, the proposed analysis for the DCS and HCS will be extended for a two-level hybrid cache system, where a root cache at the second level resolves the misses of the DCS at the first level.

Furthermore, for the DCS, it is important to extend the proposed analysis for heterogeneous caches that use different consistency mechanisms and replacement policies, and experience different access patterns. Also, I believe that the proposed analytical model can be extended for other DCS sharing protocols, such as cache array routing protocol (CARP) [45].

Furthermore, evaluating the effect of relaxing some ICP assumptions will be interesting, for example, relaxing the assumption that the local cache is not affected by the remote hits. This is called dependent-LRU, as discussed in Section 2.6.2. Fig. 6.1 shows the Markov chain that can be used to calculate the local hit of cache $k$, $H^L_k(t)$, under TTL-IE, assuming dependent-LRU. Unlike the Markov chain shown in Fig. 3.8 for SCS, the rate at which an object $i$ moves from a cache-fresh state (CFS) to state $C_k$ in Fig. 6.1, $\omega_k(t, i)$, depends on the remote access pattern, such that

$$\omega_k(t, i) = \lambda_k(i) + \sum_{q=1,q\neq k}^K \lambda_q(i) P^*_q(t, i, 0) \quad (6.1)$$

where a cache $k$, unlike the other caches in the DCS, works under dependent-LRU. Note that $P^*_q(t, i, 0) \forall q$ has to be calculated first in order to find $H^L_k(t)$. On the other
Figure 6.1: LRU Markov chain for object \( i \) under TTL-IE, assuming FCs and dependent-LRU

hand, since the cache does not cache an object based on a remote request, the rate at which an object \( i \) moves from state 0 to state \( C_k \) remains \( \lambda_k(i) \). Hence, \( \epsilon_k^z(t, i, j) \) is calculated as follows

\[
\epsilon_k^z(t, i, j) = \sum_{m=1, m \neq i}^M \lambda_k(m) P_{k}^{z-1}(t, m, 0) + \sum_{m=1, m \neq i}^M \omega_k(t, i) \sum_{c=1}^j P_{k}^{z-1}(t, m, c) \quad (6.2)
\]

where \( j \leq C_k \).

Similarly, Fig. 6.2 shows the Markov chain used to calculate \( H_k^z(t) \) under TTL-T dependent-LRU, where \( \varsigma_k(t, i) = \eta \lambda_k(i) \), and \( \eta \) is the validation factor.

Another ICP assumption that could be relaxed is that the remote request does not initiate a validation request if the requested object on the remote cache object is expired. When relaxing this condition for a cache \( k \) in the DCS, the rate at which an expired objects in the cache \( k \) is validated increases. Hence, the rate at which an expired object \( i \) returns to the CFS \( 2C_k - 1 \) in Fig. 6.2 will be \( \varsigma_k(t, i) = \eta \omega_k(i) \).
Figure 6.2: LRU Markov chain for object $i$ under TTL-T, assuming FCs and dependent-LRU
Appendix A

Convergence of the Proposed Analysis

This appendix discusses the convergence of the proposed contraction mapping analysis introduced in Chapter 3. First, a brief background for the contraction mapping theory (CMT) is presented in Section A.1. Then, the convergence of FIFO analysis under NEM is discussed in Section A.2. The convergence of FIFO analysis under TTL-IE and TTL-T is discussed in Section A.3. Finally, the convergence of LRU analysis under NEM is discussed in Section A.4.

A.1 Contraction Mapping Theory (CMT)

The principle of contraction mapping, and the corresponding contraction mapping theorems (CMTs), have proved extremely useful in the analysis of nonlinear dynamical systems [32, 33, 34]. The most basic CMT in $\mathbb{R}$, which was published by Banach in 1922, states that

**Theorem 1 (Banach’s CMT in $\mathbb{R}$) [32]:** Consider $f : \mathbb{R} \rightarrow \mathbb{R}$ is a continues scalar-valued function such that: (1) For every $x$ from the interval $[a, b]$, its image $f(x)$ also belongs to the same interval $[a, b]$, and (2) There exists a positive constant $c < 1$, where $|f(x) - f(y)| \leq c|x - y|, \forall x, y \in [a, b]$, or $|f'(x)| \leq c, \forall x \in (a, b)$. Then $f$ is called a contraction mapping over $[a, b]$ and the equation $x = f(x)$ has a unique solution $x^* \in [a, b]$, and the iteration $x^{z+1} = f(x^z)$ converges to $x^*$, starting from any
point $x^0 \in [a, b]$. 

The first condition in Theorem 1 guarantees the existence of the solution $x^* \in [a, b]$. This can be explained using the intermediate value theorem, as follows. Since $f(a) \geq a$ and $f(b) \leq b$, we have $f(b) - b \leq 0 \leq f(a) - a$. The difference $f(x) - x$ is continuous, so by the intermediate value theorem 0 is a value of $f(x^*) - x^*$ for some $x^* \in [a, b]$, and that $x^*$ is a fixed point of $f$ [33]. The second condition in Theorem 1 guarantees the uniqueness of the fixed point (i.e. $x^*$ is the only fixed point in $[a, b]$), and the converges to $x^*$ starting from any point $x^0 \in [a, b]$.

For example, $y = \cos(x)$ and $y = x$ intersect once, as shown in Fig. A.1, which means the cosine function has a unique fixed point in $\mathbb{R}$. From Fig. A.1, this fixed point lies in $[0, 1]$ [33].

![Figure A.1: \( \cos(x) \) is a contraction mapping [33].](image)

To proof that $\cos(x)$ is a contraction mapping on $[0, 1]$, first lets check $\cos(x)$ maps $[0, 1]$ to $[0, 1]$. Since cosine is decreasing on $[0, 1]$ and $\cos(1) \approx 0.54$, $\cos([0, 1]) \rightarrow [0, 1]$. Second, it is difficult to find $c < 1$ such that $|\cos(x) - \cos(y)| \leq c|x - y|$, $\forall x, y \in [0, 1]$. However, calculating $|\cos(t)'| = |\sin(t)|$ over $[0, 1]$ will give us a contraction constant, $c$. Since $\sin(t)$ is increasing (monotonic increase function) on $[0, 1]$, $\forall t \in [0, 1]$ we have $|\sin(t)| = \sin(t) \leq \sin(1) \approx 0.84147$. Therefore $|\cos(x) - \cos(y)| \leq 0.8415|x - y|$. 

143
Thus, from Theorem 1, there is a unique solution of \( \cos(x) = x \in [0, 1] \) and it can be found by iterating the function \( \cos(x) \) starting from any initial point in \([0, 1]\). Table A.1 shows the contraction mapping iterations of \( \cos(x) \) starting from \( x^0 = 1 \):

**Table A.1: The contraction mapping of \( \cos(x) \) [33].**

<table>
<thead>
<tr>
<th>( z )</th>
<th>( x^z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5403023</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>18</td>
<td>0.7393038</td>
</tr>
<tr>
<td>19</td>
<td>0.7389377</td>
</tr>
<tr>
<td>20</td>
<td>0.7391843</td>
</tr>
<tr>
<td>21</td>
<td>0.7390182</td>
</tr>
</tbody>
</table>

Furthermore, in some situations, a function is not a contraction but an iterate of it is. This turns out to suffice to get the conclusion of the contraction mapping theorem for the original function, according to the following theorem.

**Theorem 2 (Generalizations of Theorem 1) [33]:** Consider \( f : \mathbb{R} \to \mathbb{R} \) is a continues scalar-valued function, such that some iterate \( f^z : [a, b] \to [a, b] \) is a contraction, then \( f \) has a unique fixed point, \( x^* \), which can be obtained by iteration of \( f \) starting from any \( x^0 \in [a, b] \).

For example, Fig. A.2 shows that \( y = e^x \) and \( y = x \) intersect once, so \( e^a = a \) for a unique real number \( a \). The function \( f(x) = e^x \) is not a contraction on \( \mathbb{R} \) (for instance, \(|f(-2) - f(0)| \approx 6.38 > | -2 - 0|\)), but its second iterate \( g(x) = f^2(x) = e^{-e^{-x}} \) is a contraction on \( \mathbb{R} \). For some \( t \) between \( x \) and \( y \), where \(|g'(t)| = |e^{-e^{-t}}e^{-t}| = e^{-(t+e^{-t})} \leq \)
Figure A.2: $e^x$ is a contraction mapping [33].

$e^{-1}$ (since $t + e^{-t} \geq 1, \forall t \in \mathbb{R}$). Hence $f^2$ has contraction constant $1/e \approx 0.367 < 1$. From Theorem 2, the solution to $e^{-a} = a$ can be approximated by iteration of $f$ starting with any real number like the example in Table A.1 [33].

Moreover, for a vector-valued function, the Theorem 1 can be extended to $\mathbb{R}^N$ as follows:

**Theorem 3 (CMT in $\mathbb{R}^N$) [34]:** A function $f: \mathbb{R}^N \to \mathbb{R}^N$ is a contraction mapping on a domain $\pi \subset \mathbb{R}^N$, if (1) it maps $\pi$ to itself, so $f(u) \in \pi$ whenever $u \in \pi$, and (2) there exists a constant $0 \leq c < 1$ such that $\|g(u) - g(v)\| \leq c\|u - v\|, \forall u, v \in \pi$, or if the Jacobian matrix norm\footnote{Both 1 or $\infty$ norms can be used [34]} $\|f'(u)\| < 1, \forall u \in \pi$.

The Jacobian matrix of $f'(u)$, where $u = [u_1, u_2, \ldots, u_N]$, can be calculated using (A.1), and the the infinity norm of the matrix $v = f'(u)$ is given by $\|v\|_{\infty} = $
\[ \max_{1 \leq i \leq N} (|v_i|). \]

\[ f'(u) = \begin{bmatrix}
\frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \cdots & \frac{\partial f_1}{\partial u_i} & \cdots & \frac{\partial f_1}{\partial u_N} \\
\frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \cdots & \frac{\partial f_2}{\partial u_i} & \cdots & \frac{\partial f_2}{\partial u_N} \\
\vdots & \vdots & \ddots & \vdots & \cdots & \vdots \\
\frac{\partial f_{N-1}}{\partial u_1} & \frac{\partial f_{N-1}}{\partial u_2} & \cdots & \frac{\partial f_{N-1}}{\partial u_i} & \cdots & \frac{\partial f_{N-1}}{\partial u_N} \\
\frac{\partial f_N}{\partial u_1} & \frac{\partial f_N}{\partial u_2} & \cdots & \frac{\partial f_N}{\partial u_i} & \cdots & \frac{\partial f_N}{\partial u_N}
\end{bmatrix} \tag{A.1} \]

### A.2 FIFO under NEM

In this section, the convergence of FIFO analysis under NEM is discussed. First, Section A.2.1 discusses the convergence of FIFO analysis in the steady state, assuming uniform object popularity. Then, Section A.2.2 discusses the convergence of FIFO analysis in the steady state, assuming Zipf-like object popularity. Section A.2.3 discusses the convergence of the FIFO analysis in the transient state.

#### A.2.1 Convergence in Steady State, Assuming Uniform Object Popularity

Markov chain balance equations can be used to show convergence for a scalar function under the following assumptions:

1. **(S.1)** The cache is in steady state (i.e. \( t \to \infty \)). Thus, \( P^z(i, 0) = \lim_{t \to \infty} P^z(t, i, 0) \) \( \forall i \in [1, M] \).
2. **(S.2)** The objects are requested with equal probability (i.e. \( \alpha = 0 \)). Thus, \( P^z(0) = P^z(i, 0) \) \( \forall i \in [1, M] \).

Also, in order to simplify the use of the balance equations, NEM is assumed, such
that:

\[ (S.3) \] Objects do not expire (i.e. \( \mu(i) = 0 \ \forall \ i \)).

According to Theorem 1 in Section A.1, \( f : (0, 1] \rightarrow (0, 1] \) is a contraction mapping and converges to a unique fixed point, \( P^*(0) \), starting from any initial point, \( P^0(0) \in (0, 1] \), if:

\[ (C.1) \ f : (0, 1] \rightarrow (0, 1] \text{ is a continuous function.} \]

\[ (C.2) \ | f'(P(0)) | < 1 \ \forall P(0) \in (0, 1]. \] Also, \( (C.2) \) can be generalized to:

\[ (C.2.1) \ | g'(P(0)) | < 1 \ \forall P(0) \in (0, 1], \ g(P(0)) = f(f(P(0))). \]

For FIFO, according to (3.15) and using \( (S.2) \), we have:

\[ \epsilon^z(i, j) = (\beta - \lambda)P_{z-1}(0) \] \hspace{1cm} (A.2)

Using (A.2) and \( (S.3) \) to solve the balance equations and calculate the limiting probability \( P^z(0) \), we have:

\[ P^z(0) = \frac{(M - 1)P_{z-1}(0)}{(M - 1)P_{z-1}(0) + C} = \frac{\rho P_{z-1}(0)}{\rho P_{z-1}(0) + 1} \]

where \( \rho \approx M/C > 1 \), for large \( M \), or equivalently

\[ f(P(0)) = \frac{\rho P(0)}{\rho P(0) + 1} \] \hspace{1cm} (A.3)

From (A.3), \( f(P(0)) \) satisfies \( (C.1) \). Also,

\[ f'(P(0)) = \frac{\rho}{(\rho P(0) + 1)^2} \] \hspace{1cm} (A.4)
\[ f'(f(P(0))) = \frac{\rho^2}{[\rho^2P(0) + \rho P(0) + 1]^2} \]  \hspace{1cm} (A.5)

Note that the analysis in Section 3.4.1 calculates \( P^1(0) \) assuming that all objects are not in cache (i.e., \( P^0(0) = 1 \)). Hence, the analysis initializes \( P(0) = 1 \) then \( P(0) \) decreases iteratively until reaching the fixed point \( P^*(0) = \frac{\rho - 1}{\rho} \), according to (A.3). Therefore, it is suffice to proof that the maximum slope of \( f(P_0) \), which occurs at the fixed point \( P^*(0) \), is less than 1 (i.e., \( |f'(P^*(0))| < 1 \), or \( |f'(f(P^*(0)))| < 1 \)). From (A.4), \( f(P(0)) \) satisfies (C.2) \( \forall P(0) \in [P^*(0), 1] \) even for very small \( \rho \). The example in Fig. A.3 shows that when \( \rho = 2 \), the analysis converges to \( P^*(0) = \frac{\rho - 1}{\rho} = 0.5 \) and \( f'(P^*(0)) = f'(0.5) = 0.5 < 1 \). Note that in Fig. A.3 two functions are plotted:

- \( Y = P(0) \)
- \( Y = f_1(P(0)) \), where the function \( f_1 \) is calculated using the exact expressions provided in Section 3.4.1

![Figure A.3: FIFO iterative analysis, where \( M = 100 \) and \( C = 50 \).](image)

Another example in Fig. A.4 shows that the analysis converges to \( P^*(0) = \frac{\rho - 1}{\rho} = \)
0.9 from the right, under the assumption that \( P(0) \) is initialized to 1. Thus, the output steady state hit ratio is \( C/M = 10\% \), as illustrated in Table A.2.

![Figure A.4: FIFO iterative analysis, where \( M = 1000 \) and \( C = 100 \).](image)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>( P^z(0) )</th>
<th>( f_1(P^z(0)) )</th>
<th>Hit ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.90901</td>
<td>9.09918%</td>
</tr>
<tr>
<td>2</td>
<td>0.90901</td>
<td>0.90080</td>
<td>9.91965%</td>
</tr>
<tr>
<td>3</td>
<td>0.90080</td>
<td>0.89999</td>
<td>10.0009%</td>
</tr>
<tr>
<td>4</td>
<td>0.89999</td>
<td>0.89999</td>
<td>10.0009%</td>
</tr>
</tbody>
</table>

Furthermore, according to (A.5), the analysis converges to \( P^*(0) = \frac{\varphi - 1}{\varphi} \) when \( \varphi \) is small, even when \( P(0) \) is initialized by a small value that is bigger than zero (e.g. for \( \varphi = 10 \) and \( P(0) = 0.1 \), \( f'(f(0.1)) = 0.694 < 1 \)).

### A.2.2 Convergence in Steady State, Assuming Zipf-like Object Popularity

According to (3.15), we have
\[ \epsilon^z(i) = \sum_{m=1, m \neq i}^{M} \lambda(m)P^{z-1}(m, 0) \quad (A.6) \]

Since the Markov chain is at equilibrium, we have

\[ \lambda(i)P^z(i, 0) = \epsilon^z(i)P^z(i, 1) \]
\[ \epsilon^z(i)P^z(i, 1) = \epsilon^z(i)P^z(i, 2) \]
\[ \epsilon^z(i)P^z(i, 2) = \epsilon^z(i)P^z(i, 3) \]
\[ \vdots \]
\[ \epsilon^z(i)P^z(i, j) = \epsilon^z(i)P^z(i, j+1) \]
\[ \vdots \]
\[ \epsilon^z(i)P^z(i, C-1) = \lambda(i)P^z(i, C) \]
\[ \epsilon^z(i)P^z(i, C) = \lambda(i)P^z(i, 0) \]

Thus,

\[ P^z(i, C) = P^z(i, C-1) = \ldots = P^z(i, 1) = \frac{\lambda(i)}{\epsilon^z(i)}P^z(i, 0) \quad (A.7) \]

From (A.7), and since \( \sum_{c=0}^{C} P^z(i, c) = 1 \), we have
\[ 1 = P^z(i, 0) + CP^z(i, C) \]

\[ 1 = P^z(i, 0) + \frac{C\lambda(i)}{\epsilon^z(i)} P^z(i, 0) \]

\[ 1 = P^z(i, 0) \left[ 1 + \frac{C\lambda(i)}{\epsilon^z(i)} \right] \]

\[ P^z(i, 0) = \frac{\epsilon^z(i)}{C\lambda(i) + \epsilon^z(i)} \quad (A.8) \]

From (A.8), and using (A.6), we have

\[ P^z(i, 0) = \frac{\sum_{m=1, m \neq i}^M \lambda(m)P^{z-1}(m, 0)}{C\lambda(i) + \sum_{m=1, m \neq i}^M \lambda(m)P^{z-1}(m, 0)} \quad (A.9) \]

Let \( v_i^z \) denote \( P^z(i, 0) \) and \( \mathbf{v} = [v_1, v_2, \ldots, v_M] \). Thus, \( f(\mathbf{v}) = [f_1, f_2, \ldots, f_M] \), where \( f_i = \frac{\sum_{m=1, m \neq i}^M \lambda(m)v_m^{z-1}}{C\lambda(i) + \sum_{m=1, m \neq i}^M \lambda_m v_m} \), according to (A.9). Therefore,

\[
\begin{bmatrix}
0 & \frac{\partial f_1}{\partial v_2} & \ldots & \frac{\partial f_1}{\partial v_i} & \ldots & \frac{\partial f_1}{\partial v_M} \\
\frac{\partial f_2}{\partial v_1} & 0 & \ldots & \frac{\partial f_2}{\partial v_i} & \ldots & \frac{\partial f_2}{\partial v_M} \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\frac{\partial f_{M-1}}{\partial v_1} & \frac{\partial f_{M-1}}{\partial v_2} & \ldots & \frac{\partial f_{M-1}}{\partial v_i} & \ldots & \frac{\partial f_{M-1}}{\partial v_M} \\
\frac{\partial f_M}{\partial v_1} & \frac{\partial f_M}{\partial v_2} & \ldots & \frac{\partial f_M}{\partial v_i} & \ldots & 0
\end{bmatrix}
\]

where \( \frac{\partial f_i}{\partial v_i} = 0 \). For \( x \neq i \), we have

\[
\frac{\partial f_i}{\partial v_x} = \frac{\partial}{\partial v_x} \left[ \frac{\sum_m \lambda_m v_m}{C\lambda(i) + \sum_m \lambda_m v_m} \right] \quad (A.10)
\]

For \( w \neq x \), we have
\[
\frac{\partial f_i}{\partial v_x} = \frac{\partial}{\partial v_x} \left( \lambda(x)v_x + \sum_w \lambda(w)v_w \right)
\]

\[
= \frac{\lambda(x)(\lambda(x)v_x + C\lambda(i) + \sum_w \lambda(w)v_w) - \lambda(x)(\lambda(x)v_x + \sum_w \lambda(w)v_w)}{(C\lambda(i) + \lambda(x)v_x + \sum_w \lambda(w)v_w)^2}
\]

\[
= \frac{C\lambda(i)\lambda(x)}{(C\lambda(i) + \sum_m \lambda(m)v_m)^2}
\]

\[
= \frac{C\lambda(i)\lambda(x)}{(C\lambda(i) + \pi(i))^2}
\]

where \(\pi(i) = \sum_{m \neq i} \lambda(m)v_m = \beta - \lambda(i)\). Thus, the sum of the \(i\)th row, \(y_i\), in \(f'(v)\) is

\[
y_i = \frac{C\lambda(i)\sum_m \lambda(m)}{(C\lambda(i) + \pi(i))^2}
\]

\[
= \frac{C\lambda(i)(\beta - \lambda(i))}{(C\lambda(i) + \pi(i))^2}
\]

\[
= \frac{C\lambda(i)\beta - C\lambda(i)^2}{(C\lambda(i) + \pi(i))^2}
\]

(A.11)

In order to calculate \(||f'(v)||_\infty = \max_{1 \leq i \leq M}(|y_i|)\), \(\frac{\partial y_i}{\partial \lambda(i)}\) has to be calculated first.

Since the miss rate of a single object is so small compared with the aggregate miss rates of all \(M\) other objects, it’s reasonable to assume that \(\pi\) is constant \(\forall i\). This will allow us to calculate \(\frac{\partial y_i}{\partial \lambda(i)}\) by taking the derivative of (A.11) as follows

\[
\frac{\partial y_i}{\partial \lambda(i)} = \frac{(C\lambda(i) + \pi)^2(C\beta - 2C\lambda(i)) - (C\lambda(i)\beta - C\lambda(i)^2)(2C)(C\lambda(i) + \pi)}{(C\lambda(i) + \pi)^4}
\]

\[
= \frac{(C\lambda(i) + \pi)(C\beta - 2C\lambda(i)) - (C\lambda(i)\beta - C\lambda(i)^2)(2C)}{(C\lambda(i) + \pi)^3}
\]

\[
= \frac{C^2\lambda(i)\beta - 2C^2\lambda^2 + C\pi \beta - 2C\pi \lambda(i) - 2C^2\lambda(i)\beta + 2C^2\lambda(i)^2}{(C\lambda(i) + \pi)^3}
\]

\[
= \frac{C\pi \beta - 2C^2\pi \lambda(i) - C^2\beta \lambda(i)}{(C\lambda(i) + \pi)^3}
\]

(A.12)

Then, \(\lambda(i)|_{\frac{\partial y_i}{\partial \lambda(i)} = 0}\) is calculated as
\[ \lambda(i)\bigg|_{\frac{\partial y}{\partial x(i)}} = 0 = \frac{\beta \pi}{C\beta + 2\pi} \]  \hspace{1cm} (A.13)

where \( \frac{\partial^2 y}{\partial x(i)} \) is negative for \( \frac{\beta \pi}{C\beta + 2\pi} \). Thus, \( \lambda(i)\bigg|_{\frac{\partial y}{\partial x(i)}} = 0 = \frac{\beta \pi}{C\beta + 2\pi} \) is a maximum point.

Therefore, (A.11) and (A.13) can be used to calculate \( ||f'(V)||_{\infty} \) as follows:

\[
||f'(V)||_{\infty} = \frac{C\frac{\beta \pi}{C\beta + 2\pi} - C\left(\frac{\beta \pi}{C\beta + 2\pi}\right)^2}{\left(C\frac{\beta \pi}{C\beta + 2\pi} + \pi\right)^2} = \frac{(C\beta^2\pi)(C\beta + 2\pi) - (C\beta^2\pi^2)}{(C\beta + 2\pi)^2(C\frac{\beta \pi}{C\beta + 2\pi} + \pi)^2} = \frac{(C\beta^2\pi)(C\beta + 2\pi) - (C\beta^2\pi^2)}{2C\beta\pi^2(C\beta + 2\pi) + \pi^2(C\beta + 2\pi)^2 + (C\beta\pi)^2} = \frac{4C^2\beta^2\pi^2 + 8C\beta\pi^2 + 4\pi^3}{\pi 4C^2\beta^2 + 8C\beta\pi + 4\pi^2} \hspace{1cm} (A.14)
\]

From (A.14), \( ||f'(v)||_{\infty} < 1 \), as long as \( \pi \) is bigger than \( \pi = \beta/4 \) (i.e as long as the hit ratio is lower than 75%), regardless of the cache capacity, \( C \).

### A.2.3 Convergence in Transient State

Since \( f(P(0)) \) for FIFO is a bounded monotonic function, if \( f(P(0)) \) is a contraction mapping, then I argue that any monotonic function \( g(P(0)) \) (that have the same shape of \( f(P(0)) \)) is also a contraction mapping on \((0, 1]\), if it has the following properties:

(C.3) \( g : (0, 1] \to (0, 1] \) is a continuous function.

(C.4) \( g(P(0)) \geq f(P(0)) \) \hspace{1cm} \forall P(0) \in (0, 1].\)
Hence, the following conjecture can be made: The proposed analysis is a contraction mapping in the transient state (i.e. small $t$), since $f(P(0))_{|t<\infty} \geq f(P(0))_{|t=\infty}$ \quad \forall \; P(0) \in (0,1].$

A.3 FIFO under TTL-IE and TTL-T

Similar to Section A.2.3, based on (C.3) and (C.4), the following conjecture can be made: the proposed analysis is a contraction mapping for $\mu > 0$ (assuming $\mu = \mu(i)\forall i$), since $f(P(0))|_{\mu>0} : (0,1] \rightarrow (0,1]$ and $f(P(0))|_{\mu>0} > f(P(0))|_{\mu=0} \; \forall \; P(0) \in (0,1].$

However, convergence for the general case (i.e., $\mu > 0$, $t$ is small, and $\alpha > 0$) is very difficult to prove. In the remainder of this section, this issue is discussed, assuming TTL-IE.

In the general case under TTL-IE, the goal is to show that the analysis converges to a unique fixed point $H^*(t)$, such that

$$H^*(t) = \lim_{z \to \infty} H^z(t) = \lim_{z \to \infty} \sum_{i=1}^{M} [1 - P^z(t, i, 0)] \frac{\lambda(i)}{\beta}$$

Let $\mathbf{v}^z(t)$ denote the probability vector at iteration $z$

$$\mathbf{v}^z(t) = [P^z(t, 1, 0), P^z(t, 2, 0), \ldots, P^z(t, i, 0), \ldots, P^z(t, M, 0)]$$

The input vector $\mathbf{v}^z(t)$ generates an output vector $\mathbf{v}^{z+1}(t) = f(\mathbf{v}^z(t))$
Thus, for the system to be convergent, we have to prove that the system converges to a unique input vector $v^*(t)$ that satisfies

$$
\lim_{z \to \infty} v^z(t) = v^*(t)
$$

So, we need to prove that the system converges to a unique set of transition rate matrices $Q^*(i) \forall i \in [1, M]$, that satisfies

$$
P^*(t, 1, 0) = [\exp(Q^*(1)t)]_{1,1}
$$

$$
\vdots
$$

$$
P^*(t, i, 0) = [\exp(Q^*(i)t)]_{1,1}
$$

$$
\vdots
$$

$$
P^*(t, M, 0) = [\exp(Q^*(M)t)]_{1,1}
$$

where $Q^{z+1}(i)$ is a function of a vector containing the values $P^z(t, m, 0) \forall m \in [1, M]$, $m \neq i$. According to Theorem 3 in Section A.1, the proposed system is a contraction mapping when the Jacobian matrix $\infty$ norm $||f'(V(t))||_\infty < 1$ [34]. Thus, we have to
calculate

\[
\begin{bmatrix}
\frac{\partial f_1}{\partial v_1} & \frac{\partial f_1}{\partial v_2} & \cdots & \frac{\partial f_1}{\partial v_i} & \cdots & \frac{\partial f_1}{\partial v_M} \\
\frac{\partial f_2}{\partial v_1} & \frac{\partial f_2}{\partial v_2} & \cdots & \frac{\partial f_2}{\partial v_i} & \cdots & \frac{\partial f_2}{\partial v_M} \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\frac{\partial f_{M-1}}{\partial v_1} & \frac{\partial f_{M-1}}{\partial v_2} & \cdots & \frac{\partial f_{M-1}}{\partial v_i} & \cdots & \frac{\partial f_{M-1}}{\partial v_M} \\
\frac{\partial f_M}{\partial v_1} & \frac{\partial f_M}{\partial v_2} & \cdots & \frac{\partial f_M}{\partial v_i} & \cdots & \frac{\partial f_M}{\partial v_M}
\end{bmatrix}
\]

where \( v_i \) denotes \( P(t, i, 0) \) and \( f_i = [\exp(Q(i)t)]_{1,1} \). After considerable effort, I was unable to find a suitable technique for determining the derivative of \( \frac{\partial f(P)}{\partial v_m} \), since I could not calculate \( \frac{\partial [\exp(Q,t)]_{1,1}}{\partial P_m}, m \neq i \).

A.4 Convergence of LRU under NEM

In this section, the convergence of the proposed analysis for LRU under NEM is discussed assuming that the cache is in steady state and objects are equally likely to be requested. Note that, convergence in transient state and under TTL-T or TTL-IE is not provided for the same reasons discussed in Section A.3.

According to (3.26) and using (S.2), we have:

\[
\varepsilon^z(j) = \lambda(M-1)x^{z-1}(j)
\]  

(A.15)

where \( x^z(j) = \sum_{c=0}^{j} P^z(c) \). Using (A.15) and the balance equations to calculate the limiting probabilities \( P^z(C), P^z(C-1), \) and \( P^z(0) \), we have:
\[ P^z(C) = \frac{1}{(M-1)} \sum_{j=0}^{C-1} P^z(j) = \frac{1}{M} \quad (A.16) \]

\[ P^z(C-1) = \frac{M-1}{(M-1)x^{z-1}(C-1)+1} P^z(C) \quad (A.17) \]

\[ P^z(0) = (M-1)x^{z-1}(1)P^z(1) \quad (A.18) \]

For \( P^z(j) \), where \( 0 < j < C - 1 \),

\[ P^z(j) = \frac{(M-1)x^{z-1}(j+1)}{((M-1)x^{z-1}(j)+1)} P^z(j+1) \quad (A.19) \]

Therefore, from (A.16), (A.17), (A.18), and (A.19) we have:

\[ P^z(0) = (M-1)P^z(C) \prod_{j=1}^{C-1} \frac{(M-1)x^{z-1}(j)}{[(M-1)x^{z-1}(j)+1]} \]

\[ \approx \prod_{j=1}^{C-1} \frac{Mx^{z-1}(j)}{[Mx^{z-1}(j)+1]} \quad \text{for large } M \approx M - 1 \]

\[ \approx \prod_{j=1}^{C-1} \frac{Mb^{z-1}(j)P^{z-1}(0)}{[Mb^{z-1}(j)P^{z-1}(0)+1]} \quad (A.20) \]

where \( b^{z-1}(j) = \frac{x^{z-1}(j)}{P^{z-1}(0)} \) is constant. According to (A.18) and (A.19), the summation of the probabilities \( P^{z-1}(w) \), where \( 0 < w < C \), ultimately equals a scaled version of \( P^{z-1}(0) \), which is constant. From (A.20), \( f(P(0)) \) satisfies (C.1).

Note that the difference between the \( \max(b^{z-1}(j)) = b^{z-1}(C-1) \) and \( \min(b^{z-1}(j)) = b^{z-1}(1) \) is small, especially when \( P^{z-1}(0) \) is large (or when \( \frac{M}{C} \) is large). Therefore, the
product in (A.20) that uses the true values of $b^{z-1}(j)$ can be approximated by using only one value, $\pi \in [\min(b^{z-1}(j)), \max(b^{z-1}(j))],$ such that

$$P^z(0) \approx \frac{[\pi P^{z-1}(0)]^{C-1}}{[\pi P^{z-1}(0) + 1]^{C-1}}, \quad \pi \text{ is constant} \quad (A.21)$$

From (A.21), we have:

$$f'(P(0)) = \frac{C - 1}{P(0)} \frac{[\pi P(0)]^{C-1}}{[\pi P(0) + 1]^C} \quad (A.22)$$

For large $P(0)$ (i.e., $P(0) >> \frac{1}{M}$), (A.22) can be rewritten as

$$f'(P(0)) \approx \frac{C}{M\pi P^2(0)} \quad (A.23)$$

Assuming $\pi \approx (1 + \frac{1}{P(0)})/2$, $f(P(0))$ satisfies (C.2), where (A.23) guarantees that $|f'(P(0))| < 1$ even for small $\frac{M}{\pi}$ at small $P(0)$. For example, if $\frac{M}{\pi} = 10$ and $P(0) = 0.2$, then $f'(P(0)) = 25/30 < 1.$
Appendix B

Analysis of the HCS with Few Leaf Caches

The goal of this appendix is to develop a steady state hit ratio (SSHR) expression for the root cache in the proposed HCS shown in Fig. 1.3. The proposed analysis in this appendix considers LCE sharing protocol under NEM (i.e., objects do not expire), as discussed in Section 4.2. However, unlike the proposed expression in (4.2), the SSHR expression proposed in this appendix considers a few leaf caches, such that the content of the root cache and the leaf caches are correlated. To simplify the discussion, a FIFO cache hierarchy is considered. Depending on the number of leaf caches, $K$, the leaf cache capacity, $C_L$, and the root cache capacity $C_R$, the following four cases are discussed separately:

- Case 1: $K = 1$ and $C_R = C_L$ is discussed in Section B.1.
- Case 2: $K = 1$ and $C_R > C_L$ is discussed in Section B.2.
- Case 3: $K \geq 1$ and $C_R = C_L$ is discussed in Section B.3.
- Case 4 (General Case): $K \geq 1$ and $C_R \geq C_L$ is discussed in Section B.4.

In Section B.4, the SSHR expression introduced in the FIFO analysis is also examined for a cache hierarchy using LRU. The results shows that the proposed SSHR expression can be used to find the upper-bound SSHR and the lower-bound SSHR for both FIFO and LRU.

The list of the new symbols that are used in this Appendix is shown in Table B.1.
Table B.1: List of new symbols used in Appendix B.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_g$</td>
<td>Number of tracking leaf caches, such that $K_g \in [0, K]$</td>
</tr>
<tr>
<td>$\lambda(i)_{s=0}$</td>
<td>Request Rate of object $i$, given that object $i$ is not cached in the Root Cache</td>
</tr>
<tr>
<td>$\lambda(i)_{s&gt;0}$</td>
<td>Request Rate of object $i$, given that object $i$ is cached in the Root Cache</td>
</tr>
<tr>
<td>$\lambda_{R,g}(i)$</td>
<td>Request rate of object $i$ from a tracking leaf caches to the root cache</td>
</tr>
<tr>
<td>$\lambda_{R,n}(i)$</td>
<td>Request rate of object $i$ from an independent leaf caches to the root cache</td>
</tr>
<tr>
<td>$\lambda_{R,g}(i)_{s=0}$</td>
<td>Request rate of object $i$ from a tracking leaf caches to the root cache, given that object $i$ is not cached in the Root Cache</td>
</tr>
<tr>
<td>$\lambda_{R,g}(i)_{s&gt;0}$</td>
<td>Request rate of object $i$ from a tracking leaf caches to the root cache, given that object $i$ is cached in the Root Cache</td>
</tr>
<tr>
<td>$\lambda_{R,n}(i)_{s=0}$</td>
<td>Request rate of object $i$ from an independent leaf caches to the root cache, given that object $i$ is not cached in the Root Cache</td>
</tr>
<tr>
<td>$\lambda_{R,n}(i)_{s&gt;0}$</td>
<td>Request rate of object $i$ from an independent leaf caches to the root cache, given that object $i$ is cached in the Root Cache</td>
</tr>
</tbody>
</table>

B.1 Case 1: $K = 1$ and $C_R = C_L$

In this case, it is assumed that there is only one leaf cache (i.e. $K = 1$), with a capacity equals the capacity of the root cache (i.e. $C_L = C_R$). Hence, the FIFO leaf cache and the FIFO root cache store exactly the same objects, and thus, the probability that an object $i$ is not in the leaf cache, $S_L(i)$, is equal to the probability that object $i$ is not in the root cache, $S_R(i)$ (i.e. $S_R(i) = S_L(i) \forall i \in [1, M]$). Consequently, the SSHR of the root cache, $H_R$ is equal zero, since the root cache cannot satisfy any request that was not satisfied from the leaf cache. To verify this, it is explained below why the Markov chains of the leaf cache and the root cache are identical.

Let $\lambda(i)_{R_{s=0}}$ denote the rate at which object $i$ moves from state 0 to the top of
the FIFO queue (state $C_R$) in the root cache Markov chain (This is the Markov chain of a single FIFO cache under NEM, which is illustrated in Fig. 3.4), $\lambda(i)|_{s=0}$ is equal

$$\lambda_R(i)|_{s=0} = \lambda_L(i) \quad \text{(B.1)}$$

Therefore,

$$\epsilon^z_R(i) = \sum_{m=1, m \neq i}^M \lambda_R(m)|_{s>0} P^{z-1}_{R}(m, 0)$$

$$= \sum_{m=1, m \neq i}^M \lambda_L(m) P^{z-1}_{L}(m, 0)$$

$$= \epsilon^z_L(i) \quad \text{(B.2)}$$

From B.1 and B.2, and since $C_R = C_L$, both the leaf cache and the root cache are modeled using the same Markov chain.

The overall SSHR at the root cache, $H_R$, is the percentage of users’ requests that can satisfied from the root cache. This is equal to the probability that the object is requested given that the object in cache, $\lambda_R(i)|_{s>0}/\beta_L$, multiplied by the probability of finding the object in cache, $1 - S_R(i)$, such that

$$H_R = \sum_{i=1}^M (1 - S_R(i)) \frac{\lambda_R(i)|_{s>0}}{\beta_L}$$

$$= 0 \quad \text{(B.3)}$$

where $\lambda_R(i)|_{s>0} = 0$ is the request rate for object $i$, assuming that object $i$ is in the root cache. Note that the average request rate of object $i$ at the root cache is
\[ \lambda_R(i) = \lambda_R(i)|_{s=0}S_R(i) + \lambda_R(i)|_{s>0}(1 - S_R(i)) \]
\[ = \lambda_L(i)S_L(i) \quad \text{(B.4)} \]

The result in (B.3) mean that, if an object \( i \) in the root cache, then it must be also in the leaf cache. Thus, all requests for object \( i \) will be satisfied from the leaf cache. In this case, the SSHR expression in (4.2) overestimates \( H_R \), where \( K = 1 \) and \( C_R = C_L \).

### B.2 Case 2: \( K = 1 \) and \( C_R > C_L \)

For \( C_R > C_L \), there is a probability that the object might be in the leaf cache but not in the root cache, as shown in Fig. B.1. This probability increases as \( C_L \) increases, since the time that the object spends in the leaf cache, while it is not in the root cache, increases.

Thus, if an object \( i \) is not in the root cache, its request rate from the root cache, \( \lambda_R(i)|_{s=0} \), is less than \( \lambda_L(i) \) (i.e. \( \lambda_R(i)|_{s=0} < \lambda_L(i) \)). However, this fact is ignored and \( \lambda_R(i)|_{s=0} \) is calculated using (B.1), and \( \epsilon_R(i) \) is calculated using (B.2). Also, B.3 will be written as

\[ H_R = \frac{1}{\beta_L} \sum_{i=1}^{M} (1 - S_R(i))\lambda_R(i)|_{s>0} \]
Figure B.1: Case 2: $K = 1$ and $C_R > C_L$, where there is a probability that the leaf cache has an object that does not exist in the root cache.

From (B.4), we have

$$H_R = \frac{1}{\beta_L} \sum_{i=1}^{M} \lambda_L(i)S_L(i) - \lambda_R(i)|_{s=0}S_R(i)$$

From (B.1), we have

$$H_R = \frac{1}{\beta_L} \sum_{i=1}^{M} (S_L(i) - S_R(i))\lambda_L(i)$$

(B.5)

where $S_L(i) \geq S_R(i)$.

Fig. B.2 shows the analysis (solid line) and simulation (markers) results of FIFO root cache, assuming $K = 1$ and $C_R > C_L$. Fig. B.2 shows that (B.5) accurately estimates the SSHR of the FIFO root cache, especially when $C_R$ is large with respect to $C_L$. However, note that (B.5) may underestimate $H_R$ when $C_L$ is large with respect to $C_R$ (e.g. $C_L = C_R/2$ in Fig. B.2. This is because (B.1) overestimates $\lambda_R(i)|_{s=0}$. 
B.3 Case 3: $K > 1$ and $C_R = C_L$

Since $C_R = C_L$, if an object is currently cached at root, it must be also cached at least at one leaf cache. Therefore, with respect to a single leaf cache, $(1/K)$ of the root cache does not generate a cache hits (the $K$ leaf caches are equiprobable to initiate a request). This scenario is modeled by assuming that one leaf cache will perfectly track the root cache. Hence, if an object is currently cached in the root, it must be also cached in a certain leaf cache. This cache is called the tracking leaf cache, while
this object may or may not be cached on other leaf caches. These caches are called the independent leaf caches.

The tracking leaf cache acts like the leaf cache in case of \( K = 1 \) in Section B.1. Thus, assuming the object is not cached on the root cache, the tracking leaf cache forwards the requests to the root cache with rate \( \lambda_{R,g}(i)\big|_{s=0} \) that equals

\[
\lambda_{R,g}(i)\big|_{s=0} = \lambda_L(i) \tag{B.6}
\]

where \( g \) in \( \lambda_{R,g}(i)\big|_{s=0} \) indicates that the request rate is generated from the tracking leaf. On the other hand, the independent leaf forwards the requests to the root cache with rate \( \lambda_L(i)(1 - H_L(i)) \), whether the object is cached in the root cache, or not. Thus,

\[
\lambda_{R,n}(i)\big|_{s=0} = \lambda_{R,n}(i)\big|_{s>0} = \lambda_L(i)S_L(i) \tag{B.7}
\]

where \( \lambda_{R,n}(i)\big|_{s=0} \) is the request rate from an independent leaf, assuming that the object is not cached in the root cache. Therefore, the aggregate request rate seen by the root cache (requests from one tracking cache and \((K-1)\) independent caches), assuming that the object \( i \) is not in the root cache, \( \lambda_R(i)\big|_{s=0} \), is

\[
\lambda_R(i)\big|_{s=0} = \lambda_{R,g}(i)\big|_{s=0} + (K-1)\lambda_{R,n}(i)\big|_{s=0} \tag{B.8}
\]

Also, the aggregate request rate seen by the root cache (requests from one tracking cache and \((K-1)\) independent caches), assuming that the object \( i \) is in the root
cache, $\lambda_R(i)|_{s>0}$, is

$$\lambda_R(i)|_{s>0} = (K - 1)\lambda_{R,n}(i)|_{s>0} + \lambda_{R,g}(i)|_{s>0} \tag{B.9}$$

From (B.3) and (B.9), the SSHR of the root cache can be calculated as

$$H_R = \frac{1}{K\beta L} \sum_{i=1}^{M} \lambda_R(i)|_{s>0}(1 - S_R(i))$$

$$= \frac{1}{K\beta L} \sum_{i=1}^{M} (K - 1)\lambda_{R,n}(i)|_{s>0}(1 - S_R(i)) + \lambda_{R,g}(i)|_{s>0}(1 - S_R(i))$$

From (B.7), we have

$$H_R = \frac{1}{K\beta L} \sum_{i=1}^{M} (K - 1)\lambda_L(i)(1 - S_R(i))S_L(i) + \lambda_{R,g}(i)|_{s>0}(1 - S_R(i))$$

From (B.4), we have

$$H_R = \frac{1}{K\beta L} \sum_{i=1}^{M} (K - 1)\lambda_L(i)(1 - S_R(i))S_L(i) + \lambda_{R,g}(i)|_{s=0}S_R(i)$$

From (B.6), we have

$$H_R = \frac{1}{K\beta L} \sum_{i=1}^{M} (K - 1)\lambda_L(i)(1 - S_R(i))S_L(i) + \lambda_L(i)S_L(i) - \lambda_L(i)S_R(i)$$

$$= \frac{1}{K\beta L} \sum_{i=1}^{M} (K - 1)\lambda_L(i)(1 - S_R(i))S_L(i) + \frac{1}{K\beta L} \sum_{i=1}^{M} \lambda_L(i)(S_L(i) - S_R(i)) \tag{B.10}$$

where the first term is the SSHR due to the requests from independent leaf caches, while the second term is the SSHR due to the requests from the tracking leaf cache.
Table B.2: SSHR of FIFO root cache, where $C_R = C_L$, $M = 1000$, and $\alpha = 0.8$.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$C_R$</th>
<th>Simulations</th>
<th>Analysis, $K_g = 1$</th>
<th>Analysis, $K_g = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>20</td>
<td>6.74 %</td>
<td>6.70 %</td>
<td>6.80 %</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>13 %</td>
<td>13.19 %</td>
<td>13.37 %</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>5.5 %</td>
<td>5.41 %</td>
<td>6.80 %</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>9.7 %</td>
<td>9.6 %</td>
<td>13.37 %</td>
</tr>
</tbody>
</table>

In (B.10), as $K$ increases $H_R$ increases. Thus, the upper-bound SSHR of the root cache, assuming large $K$, can be calculated by taking the limits of (B.10) when $K \to \infty$, such that

$$H_R|_{K\to\infty} = \frac{1}{\beta_L} \sum_{i=1}^{M} \lambda_L(i)(1 - S_R(i))S_L(i)$$

Which is the same expression in (4.2) where all leaf caches are assumed to be independent caches (i.e. $K_g = 0$). The results in Table B.2 show that (B.10) accurately estimate $H_R$, while (4.2) overestimates $H_R$, especially when $K$ is small.

B.4 Case 4 (General Case): $K \geq 1$ and $C_R \geq C_L$

In general case, where $K \geq 1$ and $C_R \geq C_L$, the number of the tracking leaves, $K_g$, increases as $C_R$ increases. Thus, (B.10) will be rewritten as

$$H_R = \frac{1}{K\beta_L} \sum_{i=1}^{M} (K - K_g)\lambda_L(i)(1 - S_R(i))S_L(i) + \frac{1}{K\beta_L} \sum_{i=1}^{M} K_g\lambda_L(i)(S_L(i) - S_R(i))$$

(B.11)

Also, (B.8) will be modified to
Figure B.3: SSHR (in percent) of FIFO root cache, where $K = 4$, $C_L = 20$, $M = 1000$, and $\alpha = 0.8$.

\[
\lambda_R(i)|_{s=0} = K_g \lambda_{R,g}(i)|_{s=0} + (K - K_g)\lambda_{R,n}(i)|_{s=0} \\
= K_g \lambda_L(i) + (K - K_g)\lambda_L(i)S_L(i) \quad (B.12)
\]

where (B.11) are (B.12) the general SSHR expressions that are valid for any values of $K$, $C_L$, and $C_R$. Note that (4.2) is a special case of (B.11), where $K_g = 0$ (This yields the upper-bound SSHR). Also, (B.10) is a special case of (B.11), where $K_g = 1$.

The results in Fig. B.3 show that (B.11) accurately estimates $H_R$, if $K_g = 1$ and $C_R = C_L = 20$, as discussed in Section B.3. However, it is expected that the number of tracking caches, $K_g \in [0,K]$ increases, especially when $C_R$ increases with respect to $C_L$, Fig. B.3 shows that when $K_g = 1$ and $C_R$ is large, (B.11) overestimate $H_R$. Moreover, assuming all the leaf caches are tracking leaves (i.e. $K_g = K = 4$) yields accurate $H_R$ for large $C_R$, while it underestimates $H_R$ when $C_R$ is small.

Furthermore, the results in Fig. B.3 shows that $K_g = C_R/C_L$, $1 \leq K_g \leq K$, yields the closets lower-bound for $H_R$, as shown in Fig. B.4 (the markers are simulation
Figure B.4: SSHR (in percent) of FIFO root cache, assuming $K_g = C_R/C_L$, where $K = 4$, $C_L = 20$, $M = 1000$, and $\alpha = 0.8$.

results, and the solid line is the analysis).

Figure B.5: SSHR (in percent) of LRU root cache, where $K = 4$, $C_L = 20$, $M = 1000$, and $\alpha = 0.8$.

Finally, Fig. B.5 shows the SSHR of LRU root cache, which is generated using (B.11). Like FIFO, Fig. B.5 shows that (B.11) accurately estimates $H_R$ for LRU, if $K_g = 1$ and $C_R = C_L = 20$, and overestimates $H_R$ when $K_g = 1$ and $C_R > C_L$. However, unlike FIFO, $K_g = 4$ does not yield accurate $H_R$ when $C_R$ is large. In case of LRU, the most recently requested objects in the root must exist also in leaf caches.
(i.e. $K_g \geq 1$). However, since the object state in the leaf cache changes every cache hit, there is no guarantee that $K_g > 1$ since the contents of the root cache and the leaf caches become more independent. Fig. B.5 shows that $K_g = 1$ and $K_g = 2$ yields the closest upper-bound and lower-bound for $H_R$. On the other hand, using $K_g = 0$ (i.e. using (4.2)) when $K = 4$ generates optimistic results, as discussed in Section 4.2. However, note that, as $C_R$ increases, (4.2) becomes more accurate.
Bibliography


[28] J. Shim, P. Scheuermann, and R. Vingralek, “Proxy cache algorithms: design,


no. 6, pp. 3341-3356, Nov. 2008.


[104] V. N. Padmanabhan and L. Qiu, “The content and access dynamics of a busy web site: findings and implications,” ACM. Proc. of the conf. on Applications,


[111] National Laboratory for Applied Network Research, Assessing Average Hop Count of a Wide Area Internet Packet[online]. Available:
http://www.nlanr.net/NA/Learn/wingspan.html.


USA, pp. 161-175, 2002.


