

CACHE MANAGEMENT ALGORITHMS: SINGLE AND NETWORKED CACHES

SAEID MONTAZERI SHAHTOURI

(M.Sc.), IRAN UNIVERSITY OF SCIENCE & TECHNOLOGY, IRAN

A THESIS SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE


2015

Declaration

I hereby declare that the thesis is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

A handwritten signature in black ink that reads "S. Montazeri". The signature is written in a cursive style with a dot over the 'i' and a long horizontal stroke at the bottom.

SAEID MONTAZERI SHAHTOURI

January 21st, 2015

To my lovely wife, Narges, and my sweetheart son, Mohammad Amin!

Acknowledgments

First and foremost, I would like to express my sincerest gratitude to my supervisor, Dr. Richard Ma, for his constant support, guidance and training without which this dissertation would not have been possible. He guided me through difficult times in research and also in life, and has been mentor who has always amazed me with his intelligence and creativity and above all compassion and patience.

I would like to thank my thesis committee: Professor Tay Yong Chiang and Professor Ooi Wei Tsang from SoC for taking the time to evaluate my thesis and for giving me constructive feedback during my PhD. A special acknowledgement goes to the kind and talented colleagues and friends in System and Networking Research Lab 3 who have advised and entertained me in different moments of time: Dr. Claudia Szabo, Dr. Marian Mihailescu, Dr. Cristina Carbutaru, Dr. Bogdan Marius Tudor, Dr. Le Duy Khanh, Lavanya Ramapantulu, Dumi Loghim. Special thanks goes to my close friends Sajjad Maghareh, Dr. Mostafa Rezazad, Dr. AhmadReza Pourghaderi and Dr. Saeid Arabnezhad.

Words cannot express my gratitude and love to my wife, Narges Nourieh. Her faith in me is my greatest source of inspiration and motivation. My life will not be so complete and meaningful without her and our little son, Mohammad Amin.

Last but not the least, a special thanks goes to my parents and parents-in-law, for their unconditional love and support, as always and I would like to thank my older brother and sisters for their support and helping.

Contents

Acknowledgments	III
Abstract	VIII
List of Tables	X
List of Figures	XV
List of Algorithms	XVI
1 Introduction	1
1.1 Motivation	1
1.2 Objectives and Challenges	4
1.2.1 Single Cache Managed Independently	4
1.2.2 Single Cache Managed in a Network of Caches	6
1.2.3 Coordinated Caching Schemes	6
1.3 Contribution	7
1.4 Thesis Organization	10
2 Related Work	13
2.1 Single Cache Perspective	14
2.1.1 Addressing Contention, Thrashing and Pollution	14
2.1.2 Addressing Contention and Pollution	15
2.1.3 Addressing Thrashing and Pollution	16
2.1.4 Addressing Contention	17
2.1.5 Addressing Pollution	17
2.2 Network of Caches Perspective	18
2.2.1 Coordinated Hierarchical/Distributed Web Caching	18
2.2.2 Coordinated en-route Web Caching	20
2.2.3 Coordinated Caching for ICN	23
2.3 Summary	30
2.3.1 Single Cache Perspective	30
2.3.2 Network of Caches Perspective	31
3 Two-State Cache Management Policy	35
3.1 Two-State Policy in Single Cache	36

CONTENTS

3.1.1	Description	36
3.1.2	Synthetic Workload Evaluation	38
3.1.3	Real Workload Evaluation	41
3.2	Two-State Benefits for Network of Caches	43
3.2.1	Description	43
3.2.2	Synthetic Workload Evaluation	45
3.2.3	Real Workload Evaluation	56
3.3	Popularity Changes and Reservation	60
3.3.1	Motivation	61
3.3.2	Reservation	64
3.4	Summary	65
4	Coordinated Caching Scheme	67
4.1	N-State Policy with Reservation	68
4.1.1	Main Idea and Implementation	68
4.1.2	State Number Effect on Standalone Hit Ratio	69
4.1.3	State Number Effect on Overall Hit Ratio	71
4.1.4	3-State First Cache Hit Ratio	75
4.1.5	3-State Benefits for Overall Hit Ratio	77
4.2	Coordinated N-State Scheme	80
4.2.1	Defining Path, Closeness Rank & Useless Duplicate	81
4.2.2	Design Principles	82
4.2.3	Implementation	83
4.3	Evaluation	88
4.4	Summary	88
5	CAP: Contention-Thrashing-Pollution Aware Replacement Policy	90
5.1	A Class of Policies	91
5.1.1	Cache Division	91
5.1.2	Adaptivity	92
5.1.3	Choices of Replacement Policies (R_u, R_p)	93
5.2	CAP Combination:($RND, NoOP$)	95
5.3	CAP Implementation	96
5.3.1	CAP's Implementation Discussion	96
5.3.2	Time Complexity of CAP Replacement	97
5.3.3	Upper Bound of the False Protection Probability	99
5.4	Evaluation	101
5.4.1	Synthetic Workloads	105
5.4.2	Real Workloads	107
5.5	Summary	111
6	Coordinated CAP Scheme	112
6.1	COCAP: Coordinated CAP Scheme	112

CONTENTS

6.1.1	Centralized Control and Synchronized Caches	113
6.1.2	Synchronized Caches without Centralized Control	116
6.1.3	Removing Synchronized Assumption	120
6.2	Evaluation	121
6.2.1	General Setting	121
6.2.2	Linear Topology	123
6.2.3	Cross Traffic	127
6.2.4	Binary Tree Topology	129
6.2.5	Real ISP Topology Capture by RocketFuell	136
6.3	Summary	140
7	Conclusion	141
7.1	Thesis Summary	141
7.2	Future Work	144
7.2.1	The Effect of Coordinated Schemes on Routing	144
7.2.2	Combining CO2S, CO3S and COCAP with Traffic Engineering	144
7.2.3	Enhancing the Coordinated Schemes by Considering Neigh- bour Caches	145
	Bibliography	146
A	Synthetic Workload for Two-State Policy	157
A.1	Content Number of 100	158
A.1.1	Stack Distance	158
A.1.2	The Effect on Overall Cache Hit Ratio	159
A.1.3	The First Cache Hit Ratio	161
A.2	Content Number of 1000	162
A.2.1	Stack Distance	162
A.2.2	The Effect on Overall Cache Hit Ratio	163
A.2.3	The First Cache Hit Ratio	165
A.3	Content Number of 10000	166
A.3.1	Stack Distance	166
A.3.2	The Effect on Overall Cache Hit Ratio	167
A.3.3	The First Cache Hit Ratio	169
B	Trace Based Workload for Two-State Policy	171
B.1	Stack Distance	172
B.1.1	Average	172
B.1.2	Minimum	173
B.2	Benefit for Overall Hit Ratio	174
B.2.1	RANDOM on Second Cache	174
B.2.2	LRU on Second Cache	175
B.2.3	FIFO on Second Cache	176
B.3	The First Cache Hit Ratio	177

CONTENTS

C Synthetic Workload for Three-State Policy	178
C.1 Content Number of 100	179
C.1.1 The First Cache Hit Ratio	179
C.2 Content Number of 1000	180
C.2.1 The First Cache Hit Ratio	180
C.3 Content Number of 10000	181
C.3.1 The First Cache Hit Ratio	181
D Trace Based Workload for Three-State Policy	182
D.1 The First Cache Hit Ratio	183

Abstract

Since Internet was invented, several researches have been conducted to address its limitations. One of the most important limiting factors is the inconsistency between the communication model, which works based on *location* (where), and the usage of the Internet, which values the *content* (what). *Information*, or *Content* centric network is one of the influential trends to deal with this problem. The ICN communication model works based on content name instead of IP address. Although there are different ICN proposals, all of them have the property of integrating the router and cache because current and future Internet traffic are cachable. This makes a network of caches in the scale of Internet. In an ICN network of caches, obtaining a high overall hit ratio, more important than only having a high hit ratio in a single cache, is reduced by the *filtering effect*. The filtering effect happens where a cache is managed in correlation with other caches inside a network of caches. This correlation is due to communicating among caches with the objective of serving requests. A cache filters the requests that generate cache-hits and passes the requests that generate cache-misses. These filtering and passing change the pattern of requests such that another cache is hardly able to obtain a high hit ratio from the missed requests. Therefore, an ICN networked cache policy should address the filtering effect problem in addition to obtaining a high hit ratio. Moreover, a cache policy that is able to obtain a high hit ratio and addresses the filter effect is not able to obtain a high overall hit ratio without coordination due to redundancy. Furthermore, the coordinated schemes for ICN network of caches should have lightweight communication and processing overhead because an ICN router should operate in line speed.

The objective of this thesis is to propose the light weight coordinated schemes for ICN network of caches to obtain high overall hit ratios by addressing the filtering problem and managing the redundancy. We use two different approaches to reach our goal. In the first approach, we design a cache management policy to address the filtering effect problem. Then, we improve the performance of our policy in terms of hit ratio. Finally, we introduce our coordinated scheme integrated with our improved policy. In the second approach, we design a simple caching algorithm to obtain a high hit ratio for a single cache. Then, we tackle the filtering effect through coordinating and using our findings from the first approach.

The first approach starts by proposing a lightweight cache management algorithm called *two-state policy* to address the filter effect problem. The two-state policy manages a cache such that: i) the cache obtains a high hit ratio and, ii) the missed requests from the cache can be used by other caches to obtain a high hit ratio. From an ICN network of caches perspective, the two-state policy provides the opportunity of obtaining high hit ratios for other caches by introducing a new type of filtering. From a single cache perspective, we prove that the two-state policy obtains a hit ratio same as LRU under Independent Reference Model (IRM) assumption. In addition, we improve the adaptation property of *two-state policy* for the networks with large RTTs through a reservation mechanism. However, the

two-state policy suffers from one-timer contents (pollution) and is not comparable with other policies for real workloads. To solve this issue, we generalize the two-state policy to an n-state policy that obtains hit ratio higher than two-state while the n-state inherits the advantages of two-state such as solving the filter effect problem.

We complete our first approach by proposing a coordinated protocol through piggybacking information in extra integer fields of the request and content packets. Our scheme is integrated with n-state policy and has two versions: coordinated two-state with reservation (CO2S) and coordinated three-state with reservation (CO3S). The CO2S and CO3S use the advantages of two-state (solving filtering effect and thrashing) and three-state (solving filtering effect, thrashing and pollution) and also manage the redundancy. Consequently, our schemes obtain a high overall hit ratio by achieving a high hit ratio at both edge and core routers. This leads to bringing the popular contents close to the consumers, decreasing the content download time and decreasing the redundant packet transmissions in the network. Moreover, our schemes decrease the cache eviction rate that may lead to reducing the energy consumption. CO3S obtains a higher overall hit ratio compared to CO2S. On the other hand, CO2S obtains overall hit ratio comparable with other schemes, but decreases the eviction rate up to four orders of magnitude.

Our second approach starts by introducing a cache management policy, CAP, which addresses all of the caching problems for a single cache and is the base for a new coordinated scheme called COCAP. We introduce a class of replacement policies by dividing the cache into two variable sized segments managed independently. Among all of the combinations of the applicable replacement policies for the two segments, the combination of Random policy for missed contents and no-operation for hit contents solves all of the caching problems for a single cache while its overhead is not considerable. In addition, our trace-based simulations show that CAP obtains the hit ratio pretty close to the state-of-the-art for a single cache. Furthermore, the time complexity of CAP is constant and it does not impose memory overhead.

Our second approach is completed by proposing a coordinated scheme based on CAP, COCAP, for an ICN network of caches. The COCAP, implemented through piggybacking information in extra integer fields, solves thrashing and pollution problems using CAP and tackles the filter effect through coordination. The COCAP coordination has two characteristics: i) using freezing/updating idea of two-state ii) managing the network of caches as a virtual cache. The idea of virtual cache enables COCAP to bring the popular content close to the consumers and manage the redundancy.

List of Tables

2.1	Related work summary - single cache perspective	30
3.1	The specifications of the traces	41

List of Figures

2.1	System model for coordinated en-route caching	20
2.2	Different meta algorithms	22
3.1	State diagram of two-state cache management	36
3.2	Simple topology for filter effect experiment	40
3.3	The first cache hit ratio with different policies versus cache size	40
3.4	The first cache hit ratio with different policies versus popularity (α)	41
3.5	The first cache hit ratio with different policies versus cache size with uc trace	42
3.6	The first cache hit ratio with different policies versus cache size with pa trace	42
3.7	Minimum stack distances of the missed requests from the first cache managed by three replacement policies and two-state policy	47
3.8	Maximum stack distances of the missed requests from the first cache managed by three replacement policies and two-state policy	48
3.9	Average stack distances of the missed requests from the first cache managed by three replacement policies and two-state policy	49
3.10	Minimum stack distances of the missed requests from the first cache versus different α (Zipf slope)	50
3.11	Maximum stack distances of the missed requests from the first cache versus different α (Zipf slope)	51
3.12	Average stack distances of the missed requests from the first cache versus different α (Zipf slope)	52
3.13	The CDF plot of the stack distance from the missed requests with $\alpha = 1.2$	52
3.14	The CDF plot of the stack distance from the missed requests with $\alpha = 1.5$	53
3.15	The hit ratio of the second cache that is managed by RND. The first cache is managed by RND, LRU, FIFO and two-state policy	54
3.16	The hit ratio of the second cache that is managed by LRU. The first cache is managed by RND, LRU, FIFO and two-state policy	54
3.17	The hit ratio of the second cache that is managed by FIFO. The first cache is managed by RND, LRU, FIFO and two-state policy	55
3.18	The hit ratio of the second cache that is managed by RND. The first cache is managed by RND, LRU, FIFO and two-state policy	56

LIST OF TABLES

3.19	The hit ratio of the second cache that is managed by LRU. The first cache is managed by RND, LRU, FIFO and two-state policy	56
3.20	The hit ratio of the second cache that is managed by FIFO. The first cache is managed by RND, LRU, FIFO and two-state policy	57
3.21	The minimum stack distances of the missed requests from the first cache with trace uc	57
3.22	The maximum stack distances of the missed requests from the first cache with trace uc	58
3.23	The average stack distances of the missed requests from the first cache with trace uc	59
3.24	The hit ratio of the second cache that is managed by RND. The first cache is managed by RND, LRU, FIFO and two-state policy	60
3.25	The hit ratio of the second cache that is managed by LRU. The first cache is managed by RND, LRU, FIFO and two-state policy	60
3.26	The hit ratio of the second cache that is managed by FIFO. The first cache is managed by RND, LRU, FIFO and two-state policy	61
3.27	RTT between sending a missed request and receiving its content	62
3.28	The hit ratio of the cache with arrival rate of 10000 requests per second, updating period of 10 sec, the average popularity change period of 50 sec ($\lambda_c = 0.02$) and the probability of popularity changes, p_s , of 0.05 - C=10, N=1000, $\alpha = 1$	63
3.29	Different combinations of RTT and characteristic time that plays role in decreasing the hit ratio of LRU	63
4.1	The cache state diagram in n-state policy	69
4.2	The first cache hit ratio with different policies versus cache size	70
4.3	The first cache hit ratio with n-state versus number of states by bo2 trace	71
4.4	The first cache hit ratio with n-state versus number of states by sd trace	71
4.5	The second cache hit ratio with LRU versus number of states with $N = 1000$, $\alpha = 1$	72
4.6	The second cache hit ratio with RND versus number of states with $N = 1000$, $\alpha = 1$	72
4.7	The second cache hit ratio with FIFO versus number of states with $N = 1000$, $\alpha = 1$	73
4.8	The second cache hit ratio with LRU versus number of states by sd trace	74
4.9	The second cache hit ratio with RND versus number of states by sd trace	74
4.10	The second cache hit ratio with FIFO versus number of state by sd trace	75
4.11	The first cache hit ratio with different policies versus cache size	76
4.12	The first cache hit ratio with different policies versus α	76

LIST OF TABLES

4.13	The first cache hit ratio with different policies and traces	77
4.14	The second cache hit ratio with LRU, $N = 1000$, $\alpha = 1$	78
4.15	The second cache hit ratio with LRU, $N = 1000$, $C = 10$	78
4.16	The second cache hit ratio with LRU and different traces	79
4.17	The second cache hit ratio with RND and different traces	80
4.18	Definition of closeness rank based on path	81
4.19	Managing the duplicate copies and multipath routing	86
4.20	A linear topology with cross traffic	87
5.1	Protected and unprotected segments. $S_p = 5$ and $C = 8$	91
5.2	A cache <i>round</i> starts from 5.2a and ends with reaching 5.2e. $S_p^{max} = 5$ in this sample. The two intermediate cache states from 5.2c to 5.2d are omitted.	92
5.3	Cache hit ratio versus cache size under synthetic workload. Total number of contents in the system is 1000.	102
5.4	Cache hit ratio versus cache size under synthetic workload. Total number of contents in the system is 1000, thrashing length (β) is 100 with different portion of thrashing workload (γ).	103
5.5	Cache hit ratio versus cache size under synthetic workload. Total number of contents in the system is 1000, thrashing length (β) is 200 with different portion of thrashing workload (γ).	104
5.6	Cache hit ratio versus cache size under synthetic workload. Total number of contents in the system is 1000, thrashing length (β) is 200, portion of thrashing workload (γ) is 0.25 with different portion of pollution workload(Θ).	106
5.7	Cache hit ratio versus cache size under real workloads of hm0, hm1 and msd0.	108
5.8	Cache hit ratio versus cache size under real workloads of prn0, mds1 and proj0.	109
5.9	Cache hit ratio versus cache size under real workloads of prxy0, rsrch0 and rsrch2.	110
6.1	A single big cache	113
6.2	A big cache distributed in the network with a centralized control	114
6.3	A virtual cache with size of 3	115
6.4	The virtual cache with respect to different paths	116
6.5	A linear topology with one-way traffic	123
6.6	Overall network hit ratio	124
6.7	Average hit ratio of edge routers	125
6.8	Average hit ratio of core routers	126
6.9	Overall network hit ratio	126
6.10	Average hit ratio of edge routers	127
6.11	Average hit ratio of core routers	127
6.12	A linear topology with cross traffic	128

LIST OF TABLES

6.13	Individual hit ratio of P_1 with cache size of 10 and catalog size of 1000	128
6.14	Individual hit ratio of P_2 with cache size of 10 and catalog size of 1000	129
6.15	The overall hit ratio versus cache size for catalog size of 1000	130
6.16	The binary tree topology	130
6.17	Overall network hit ratio	131
6.18	Average hit ratio of edge routers	132
6.19	Average hit ratio of core routers	132
6.20	Average content download time	133
6.21	Average eviction rate per slot	133
6.22	Overall network hit ratio	134
6.23	Average hit ratio of edge routers	134
6.24	Average hit ratio of core routers	135
6.25	Average content download time	135
6.26	Average eviction rate per slot	135
6.27	Topology captured by rocketfuel with ISP number of 6461	136
6.28	Overall network hit ratio	137
6.29	Average hit ratio of edge routers	138
6.30	Average hit ratio of core routers	138
6.31	Average content download time	139
6.32	Transferred packet reduction ratio	139
6.33	Average eviction rate per slot	140
A.1	Average stack distance with different α , $N=100$	158
A.2	Average stack distance with different cache sizes, $N=100$	158
A.3	Minimum stack distance with different α , $N=100$	159
A.4	Minimum stack distance with different cache sizes, $N=100$	159
A.5	Second cache (FIFO) hit ratio with different α , $N=100$	159
A.6	Second cache (FIFO) hit ratio with different cache sizes, $N=100$	160
A.7	Second cache (LRU) hit ratio with different α , $N=100$	160
A.8	Second cache (LRU) hit ratio with different cache sizes, $N=100$	160
A.9	Second cache (RND) hit ratio with different α , $N=100$	161
A.10	Second cache (RND) hit ratio with different cache sizes, $N=100$	161
A.11	First cache hit ratio with different α , $N = 100$	161
A.12	First cache hit ratio with different cache sizes, $N=100$	162
A.13	Average stack distance with different α , $N=1000$	162
A.14	Average stack distance with different cache sizes, $N=1000$	162
A.15	Minimum stack distance with different α , $N=1000$	163
A.16	Minimum stack distance with different cache sizes, $N=1000$	163
A.17	Second cache (FIFO) hit ratio with different α , $N=1000$	163
A.18	Second cache (FIFO) hit ratio with different cache sizes, $N=1000$	164
A.19	Second cache (LRU) hit ratio with different α , $N=1000$	164
A.20	Second cache (LRU) hit ratio with different cache sizes, $N=1000$	164

LIST OF TABLES

A.21 Second cache (RND) hit ratio with different α , $N=1000$ 165

A.22 Second cache (RND) hit ratio with different cache sizes, $N=1000$. . 165

A.23 First cache hit ratio with different α , $N = 1000$ 165

A.24 First cache hit ratio with different cache sizes, $N=1000$ 166

A.25 Average stack distance with different α , $N=100$ 166

A.26 Average stack distance with different cache sizes, $N=1000$ 166

A.27 Minimum stack distance with different α , $N=10000$ 167

A.28 Minimum stack distance with different cache sizes, $N=10000$ 167

A.29 Second cache (FIFO) hit ratio with different α , $N=10000$ 167

A.30 Second cache (FIFO) hit ratio with different cache sizes, $N=10000$. 168

A.31 Second cache (LRU) hit ratio with different α , $N=10000$ 168

A.32 Second cache (LRU) hit ratio with different cache sizes, $N=10000$. 168

A.33 Second cache (RND) hit ratio with different α , $N=10000$ 169

A.34 Second cache (RND) hit ratio with different cache sizes, $N=10000$. 169

A.35 First cache hit ratio with different α , $N = 10000$ 169

A.36 First cache hit ratio with different cache sizes, $N=10000$ 170

B.1 Average stack distance with different traces 172

B.2 Minimum stack distance with different traces 173

B.3 Second cache (RND) hit ratio with different traces 174

B.4 Second cache (LRU) hit ratio with different traces 175

B.5 Second cache (FIFO) hit ratio with different traces 176

B.6 First cache hit ratio with different traces 177

C.1 First cache hit ratio with different α , $N = 100$ 179

C.2 First cache hit ratio with different cache sizes, $N=100$ 179

C.3 First cache hit ratio with different α , $N = 1000$ 180

C.4 First cache hit ratio with different cache sizes, $N=1000$ 180

C.5 First cache hit ratio with different α , $N = 10000$ 181

C.6 First cache hit ratio with different cache sizes, $N=10000$ 181

D.1 First cache hit ratio with different traces 183

List of Algorithms

1	CAP(R_u, R_p)	94
2	CAP($RND, NoOP$)	98

Chapter 1

Introduction

1.1 Motivation

In the beginning of Internet era, the key goal of using the Internet was resource sharing. Considering the goal, the communication model chosen for the earliest Internet was based on a conversational model between two entities. Although the Internet has received continuous improvement during the last 50 years, the communication model roughly remains the same. However, the objective of using the Internet has gradually changed from a pure resource sharing tool to a massive disseminating *information*¹ or *content* media [36]. This trend shows that the **location** (where) is now less important for users than the **content** (what) [40] but the location still has its main role in the communication model.

This difference between the communication model and the usage of the Internet causes some challenges include, but not limited to, the lack of a content distribution method in network layer and the overhead to the network such as the translation from content to location. Therefore, an efficient and scalable Internet architecture for future demand is required. The *Information* or *Content* centric network is one of the influential trends for the future Internet architecture. Information Centric Network (ICN) has the potential to cope with many architectural

¹We use terms content and information in this text interchangeably

Internet challenges because its communication model is consistent with the Internet usage. That is, the communication model works based on the content rather than the location. Although there are different ICN proposals such as NDN [40], NetInf [1] and DONA [47], integrating cache and router is common among all of them. This makes the Internet as a network of caches that is defined as a collection of caches connected to each other. Each cache either answers a request for a specific content with that content or forwards the request to the next router through the path towards the producer.

The motivation for using a network of caches in ICN is that the Internet traffic is cachable [3]. Anand et al. [3] show that caching the Internet traffic for 10 seconds can lead up to 50% hit ratio. In addition, the projections about the future [22] show that in 2016 around 86% of the customer traffic type will be video and the access pattern of video contents follows the Zipf distribution [14, 21]. Therefore, in addition to the currently cachable Internet traffic, it is highly probable that the major type of Internet traffic will be cachable. Consequently, using a network of caches in ICN has many potential benefits such as bringing the contents closer to the consumers, reducing the load on producers and decreasing the unnecessary retransmission in ISPs.

Although the usage of network of caches in ICN is promising, the efficiency of a network of caches is affected by the *filter effect* phenomenon [4, 88, 89]. A cache can be considered as a filter. That is, the cache serves the requests that generate cache-hits and forwards the requests that generate cache-misses. These filtering and passing change the pattern of requests such that subsequent caches are unable to obtain high hit ratios from the forwarded requests. The filter effect has been studied for several years from both the frequency perspective [29, 87, 88] and the time perspective [8]. To reduce the filter effect, Busari and Williamson [88] proposed to use heterogeneous replacement policies in a network of caches. Later, Ari *et al.* [4] proposed Adaptive Caching using Multiple Experts (ACME) that uses neural networks to find the optimal combination of replacement policies.

Even though the previous studies combined different replacement policies to obtain a higher hit ratio in the core routers, their results show that the filter effect still appears because the request pattern in the core routers is changed by the filtering at the edge routers. Therefore, the overall hit ratio of the network of caches is degraded. Consequently, to obtain a high overall hit ratio for a network of caches, a single cache should be managed such that it simultaneously achieves two objectives: i) obtaining a high hit ratio and ii) generating a missed request stream that can be used by other caches to obtain a high hit ratio. Although these two objectives are necessary, they are not sufficient for obtaining a high overall hit ratio in the network of caches.

In addition to the filter effect, the redundancy degrades the overall hit ratio in the network of caches. The redundancy happens when several caches keep the same copy of a content where these caches can communicate to each other and obtain a content from one of the caches. This content redundancy is the result of independent cache management algorithms that can be addressed by coordinated caching schemes. The coordinated caching scheme is defined as the way that different caches coordinate to prevent the content redundancy. There are two types of coordinated caching scheme: explicit and implicit [72]. The explicit coordination occurs when the caches share their states (or state summaries) with each other [82]. Each cache uses the state information of other caches to decide about the content that is going to be cached or evicted but the communication cost of the state exchanging is not negligible. Although explicit coordination can be considered as the method to overcome the filter effect, it is not applicable for ICN proposals due to the high communication overhead. In contrast to explicit coordination, implicit coordination may use a combination of local cache information, the cache position, and small piggybacking information exchanged by requests or content packets. Although implicit coordination is applicable for ICN, it is suffering from filter effect. Consequently, they cannot effectively increase the overall hit ratio.

1.2 Objectives and Challenges

For more than several decades, many works have been conducted to improve the cache management algorithms. However, using a network of caches is pushing the cache management algorithms to consider an extra goal to solve filter effect problem in addition to a obtaining high hit ratio. Recently, usage of network of caches in ICN has pushed new efforts to deal with the filter effect by considering the characteristics of ICN. The objective of this thesis is to propose and evaluate implicit coordinated schemes, applicable to ICN, to obtain high overall hit ratios by addressing the filtering problem and managing the redundancy.

In this section, we explain the challenges of the ICN cache management algorithms from three perspectives. Firstly, we explain the challenges from a single cache perspective that exists for a standalone cache in an ICN network of caches. Secondly, we explain the challenges that should be addressed when a cache is managed as a member of a network of caches. Finally, we explain the challenges that coordinated caching scheme, applicable to ICN, should address. After explaining all of the challenges, we introduce two approaches for tackling the challenges.

1.2.1 Single Cache Managed Independently

A single cache that is managed independently from other caches should address these five challenges:

1. **Time complexity** of a cache management policy for an ICN network of caches should be considered as an important factor. For example, Least Frequently Used (LFU), which is the optimal replacement policy under Independent Reference Model (IRM) assumption [23], has the time complexity of $O(\log n)$ where n is the number of total contents in the network. Its time complexity makes this algorithm impractical for ICN, where each router should operate at line speed [5].
2. **Pollution** occurs in the network traffic [55, 86] when the contents with low

reuse evict the contents with high reuse. This can happen when a sequence of one-timer contents is generated and these one-timer contents evict the popular contents that are regularly requested. This problem is also called scan problem in the literature.

3. **Thrashing** is defined as the situation when a working set is sequentially and repeatedly requested and is greater than the cache size. This causes continuous content evictions without any gain. For example, suppose that the workload of requests for contents a , b and c in the format of $abcabcabcabc\dots abc = (abc)^*$ comes to a Least Recently Used (LRU) cache with space for two contents. Then, the number of hits will be zero because the different contents kick each other out of the cache. Due to the large video file size, larger in the future, the thrashing could happen in the network traffic where a group of consumers requests a stream of packets repeatedly and the cache size is small (maximum of 10GB[5]) compared to large high-definition video files.
4. **Contention** is possible in an ICN router where all of interfaces share a cache. Specifically, the contention happens when all the requests in different interfaces should be serialized behind a global cache lock because of the cache management algorithm. For example, LRU requires all the hit requests (missed contents) to promote (to be written) their corresponding contents to the Most Recently Used position. To properly implement LRU, accessing the Most Recently Used (MRU) position should be in a critical section. It should be mentioned that the contention cannot be removed since several interfaces try to access to a common place. However, the granularity of the contention can be decreased from cache level to slot level. In this thesis, **solving the contention problem** is defined as decreasing the granularity of contention from the whole cache to one slot. For example, CLOCK [24], an approximation of LRU, solves the contention problem. That is, there is no need for a lock in cache level but a cache slot should be locked for writing a content or setting the reference bit.

5. **Memory overhead.** There are algorithms that keep additional information to address some of the above mentioned challenges. Although the information is limited to the meta-data of contents, the memory overhead is not negligible for caches with large number of slots.

1.2.2 Single Cache Managed in a Network of Caches

The cache algorithm, managing a single cache in a network of caches, should address one challenge.

6. **Filtering effect.** A cache can be considered a filter because it serves the requests that generate cache-hits and forwards the requests that generate cache-misses. These filtering and forwarding change the pattern of requests such that the subsequent caches hardly are able to obtain high hit ratios from the forwarded requests.

1.2.3 Coordinated Caching Schemes

The coordinated caching scheme designed for ICN should be able to address three challenges:

7. **Redundancy.** Some schemes impose redundant copies of content and these redundant copies decrease the efficiency of network of caches. Managing the redundancy is important in ICN because the cache size compared to the catalogue size (total number of contents) is very small and redundant copies can degrade the overall hit ratio drastically.
8. **Communication overhead** is high for the explicit coordination where each cache sends its state to its neighbors. Especially, the communication overhead is increased when the number of total contents in the network is much larger than the individual cache size that is the case for ICN [73] due to limitations of the current memory technologies [5]. In such case, the rate

of state change in a cache is high and consequently the amount of updating information exchanged among caches is considerable.

9. **Cross traffic** means that the traffic traverses in two or more different directions. A linear topology in which the consumers are at one end and the producers are at the other end does not have cross traffic. However, if there are consumers and producers located at both ends, there exists cross traffic when consumers at each end request the content on the other end. Some schemes only perform well when traffic moves in one direction, but the schemes for the ICN network of caches should consider the cross traffic.

We use two different approaches to reach our goal. In the first approach, we design a cache management policy to address the first six challenges including the filtering effect through introducing a cache management policy. Moreover, we manage the redundancy through implicit coordination integrated with our management policy. In the second approach, we design a simple caching algorithm to obtain a high hit ratio by tackling the first five challenges for a single cache. In addition, we tackle the filtering effect problem and redundancy through coordinating among caches and combining the finding of the first approach.

1.3 Contribution

The contributions of this thesis are as follow:

1. **Two-State: a new cache management policy.** The *two-state policy* is a new cache management policy that fetches the contents for the first C different requests where C is the number of cache slots. Then, the cache slots are frozen for a predefined period of time. During this period of time, the cache slots do not get replaced. By finishing the period, the cache repeats the process to adapt to the traffic pattern changes. This policy tackles the first six challenges except the pollution. The most important characteristic

of the *two-state policy* for ICN network of caches is to fight against the filtering effect problem by introducing a new type of filtering effect while its implementation is simple. Using the *two-state policy* instead of replacement policy at the edge router caches (directly connected to consumers) leads to higher hit ratios at the core router caches (indirectly connected to consumers). Furthermore, we mathematically prove that the *two-state policy* and LRU have the same hit ratios under the IRM assumption. Finally, we improve the adaptation property of *two-state policy* for network with large RTTs through reservation mechanism.

2. **N-State: generalization of two-state.** The n-state policy is the generalization of the two-state policy. The policy obtains higher hit ratio than the two-state by capturing the popular contents and solving the pollution problem of the two-state. We show that the improvement of cache hit ratio is considerable by increasing n from two to three, but it is not considerable for $n > 3$. Under IRM assumption, our evaluations show that the three-state policy obtains the hit ratio close to the hit ratio of LFU that is the optimal policy for IRM [23]. Moreover, we evaluate the three-state policy through real workloads and show that it obtains a high hit ratio for single cache and provides the opportunity for subsequent caches to obtain a high hit ratio too.
3. **Coordinated Scheme Integrated with N-State.** Our scheme has a simple implementation and does not impose overheads such as measuring content popularity [59] or exchanging the cache states among neighbors [37]. The coordination among ICN routers is done by piggybacking information through integer fields in the request and content packets. Our coordinated scheme is integrated with the n-state policy to obtain its advantages. Moreover, our scheme manages the redundancy and brings the popular content close to the consumers. We evaluate coordinated three-state with reservation (CO3S) and coordinated two-state with reservation (CO2S) through

synthetic and real topologies. Although our schemes obtain the high hit ratio at both edge and core routers, CO3S outperforms CO2S in terms of the overall hit ratio, content download time and transferred bytes. The CO3S improves the overall hit ratio up to seven times for small cache sizes that are important in the ICN and up to 25% for large cache sizes compared to LRU universal caching (LRU in all caches). This leads to 7% to 13% more reduction in content download time and 24% more reduction in transferring packet by CO3S. Moreover, CO2S obtains the comparable performance with other coordinated schemes while it decreases the evictions rate up to four orders of magnitude. This may have the implication of reducing the energy consumption by ICN routers.

4. **CAP: a new cache management algorithm.** We propose a new cache replacement policy called CAP to simultaneously address the first five problems. We divide the cache into two variable sized segments: *protected* and *unprotected*. The missed contents are written into unprotected segment and they are moved into protected segment if they get at least one hit before being evicted. Each segment is managed by an independent replacement policy. We explain the advantages and disadvantages of different replacement policy combinations. Finally, we choose the Random replacement policy (RND) for the unprotected segment and do nothing (no action for a content hit) for the protected segment. This combination can overcome the contention and thrashing problems. Moreover, having separate segments for protected and unprotected contents decreases the effect of the pollution problem since the one-timer contents cannot drastically affect the popular contents in the protected segment. Finally, the time complexity of CAP is constant and it does not impose memory overhead. Our evaluation through both synthetic and real workloads shows that CAP obtains the performance close to the state-of-the-art, ARC [57], in terms of cache hit ratio while ARC is prone for contention.

5. **COCAP: coordinated CAP** Obtaining a high hit ratio close to the hit ratio of the best policies for a single cache motivates us to propose a coordinated scheme based on CAP for ICN network of caches. By introducing the concept of virtual cache, COCAP caches the popular contents on the edge router and provides the opportunity for core routers to obtain a high hit ratio by using the idea of freezing the caches. For the binary tree topology without cross traffic the COCAP outperforms the CO3S and CO2S. Moreover, COCAP obtains overall hit ratio close to the CO3S for real topology with cross traffic but the CO3S obtains better performance than COCAP in terms of content download time and the traffic reduction ratio.

1.4 Thesis Organization

The thesis is organized as follows:

Chapter 2. Related Work

The second chapter presents the related work from two different perspectives: i) single cache ii) coordinated caching scheme. The related work from a single cache perspective is classified based on the number of addressing challenges i.e., pollution, contention and thrashing. Moreover, we discuss the time complexity and memory overhead for each work. Later, the related work of the coordinated scheme is classified into three categories: i) hierarchical/distributed web caching, ii) the coordinated en-route caching, iii) the recent work for ICN coordinated caching. We conclude the chapter with three tables summarizing the related work and the number of challenges that can be addressed by each algorithm or scheme.

Chapter 3. Two-State Cache Management Policy

The chapter explains our two-state cache management policy and follows by introducing the main advantage of the two-state policy. To compare two-state

with replacement policies, we represent the replacement policy by RND, LRU and FIFO applicable in an ICN router [5]. We compare two-state with a replacement policy from two different perspectives: i) single cache hit ratio ii) the effect on the hit ratio of other caches (overall hit ratio). In addition to measuring the overall hit ratio, we use stack distance [56] to quantify the effect of one cache on the network of caches. Using stack distance, we explain why two-state leads to higher overall hit ratio. In our evaluation, we use both synthetic and real workloads. The real traces, also used in Chapter 4, are analyzed in this chapter from different perspectives such as the portion of one-timer requests and one-timer contents. The less hit ratio of two-state compared to other replacement policies under realistic workloads is due to pollution problem but is addressed in Chapter 4 by introducing the n-state policy. Lastly, the chapter shows that the two-state hit ratio drop drastically when the RTTs is large. We discuss the reason for this observation and explain that this is due to inability of two-state to adapt to popularity changes for large RTTs. Finally, we explain our reservation mechanism.

Chapter 4. Coordinated Caching Scheme (CO2S and CO3S)

The chapter extends our two-state with reservation policy to n-state with reservation to obtain a higher hit ratio compared to two-state. After discussing about n-state, the chapter explains our coordinated caching scheme integrated with n-state policy in three steps. First, we introduce three concepts i) path, ii) closeness rank and iii) useless redundant copy. Then, we introduce our two design principles and high level idea through three concepts. Finally, we explain the implementation of our scheme. The evaluation of CO3S and CO2S is integrated with the evaluation of COCAP in Chapter 6.

Chapter 5. CAP Policy

This chapter introduces a class of replacement policies. We explain how different combinations obtain the advantages of three applicable replacement policies

for ICN routers [5]: FIFO, RND and LRU. Then, we discuss about our combination and explain how it can solve three caching problems, contention, thrashing and pollution, at the same time and achieve a hit ratio comparable to other state-of-the-art work. In addition, we prove that the average memory access in the worst case is $O(1)$. Finally, we evaluate CAP using synthetic and real workloads.

Chapter 6. Coordinated CAP (COCAP)

In this chapter, we present our coordinated caching scheme based on the CAP introduced in Chapter 5. First, we explain the main idea of extending the CAP for a network of caches by making two unrealistic assumptions. Then, in two steps, we remove the assumptions and explain how our main idea can be implemented through four extra fields in the request and content packets and two variables in each router. Finally, we evaluate CO2S, CO3S and COCAP using synthetic and real topologies.

Chapter 7. Conclusion

This chapter concludes the thesis and discusses the future research directions.

Chapter 2

Related Work

In this chapter, we present the related work from two different perspectives. Firstly, we describe the related work conducted for improving the performance of a standalone cache. The standalone related work is classified based on the number of addressed challenges from contention, pollution and thrashing described in Section 1.2.1. Secondly, we describe the works conducted to improve the performance of network of caches. The related work in this part is classified into three categories based on their types of network of caches. The first category describes the work for hierarchical/distributed web caching which is defined as a collection of web caches in a hierarchical/distributed arrangement. The caches coordinate to get better performance in terms of overall hit ratio. The second category includes the work for coordinated en-route caching. In en-route caching, each router has its own cache to keep the passing content for future re-references. The last category covers the recent works for ICN coordinated caching. This chapter is concluded with three tables summarizing the related work of single and network of caches. Each table highlights the research gaps from its perspective.

2.1 Single Cache Perspective

We categorize the cache management algorithms based on the number of challenges that an algorithm can simultaneously tackle. The challenges are: contention, thrashing and pollution. In addition, we mention the time complexity and memory overhead of each algorithm.

2.1.1 Addressing Contention, Thrashing and Pollution

Random (RND) replacement policy randomly evicts one content for a miss and does nothing for a hit. The RND obtains a higher hit ratio than LRU and FIFO (zero) in the presence of thrashing. RND does not suffer from the contention problem. However, the pollution affects the hit ratio of RND and its overall hit ratio is not comparable with LRU. We will explain how our CAP uses the advantages of RND and avoid its disadvantages. Moreover, the time complexity of RND is $O(1)$ and it does not impose any memory overhead.

CAR [9] combines the ARC [57] and CLOCK [24]. CAR has four doubly linked lists B1, B2, T1, and T2. B1 and B2 are simple LRU lists while T1 and T2 are CLOCKs. B1 (B2) is the meta-data for the recent (frequent) evicted contents while T1 (T2) maintains the recent (frequent) cached contents. T1 and T2 together keeps C (cache size) contents in the cache and size of each one adaptively is changed based on the workload changes. CLOCK-Pro [41] works based on reuse distance for content replacement decision. This policy categorizes the contents into cold and hot. The cold contents have large reuse distances while the hot contents have small reuse distances. The cache size, C , is adaptively divided between cold contents and hot contents. In addition, the meta-data for C evicted contents are also maintained in the memory. The policy makes an ordered list based on content access to maintain all the accessed contents (hot and cold). Every cold content which is accepted into the list should pass test period. This provides the opportunity for the cold contents to turn into a hot content by being accessed

once. Otherwise, the cold content is removed from the list. It is possible for a cold content to be removed from the memory but its meta-data is kept in the list for the test purpose.

Both CAR and CLOCK-Pro can solve the contention problem but they keep extra meta-data of $2C$ and C evicted contents respectively. This makes the overhead of CAR non negligible for the systems which need to cache a large number of small objects. In contrast, our CAP policy can solve contention problem and hit contents and does not need any meta-data about the evicted contents.

Two other techniques to reduce the impact of contention in caches are batching and pre-fetching [26]. By batching, if a cache hit happens, the replacement policy does not change its data structure. Instead, the policy appends the request into a FIFO queue and applies the corresponding changes when number of requests reaches a threshold. By pre-fetching, the required data in the critical section is read immediately before a request for lock by the replacement algorithm. Since these techniques do not need any specific requirement from the replacement policy side, they can be combined with our replacement policy.

Static caching [83] requires real-time measurements of the access frequencies for the content and this overhead might make it impractical for ICN routers to implement because of ICN router processing limits.

LRFU [50] subsumes LRU and LFU. LRFU deals with three problems based on one parameter. The parameter can convert LRFU to LRU by valuing the unprotected. On the other hand, the parameter can also convert LRFU to LFU by valuing protected. However, the time complexity of LRFU varies from constant per requests to logarithmic in cache size per request. Our policy, CAP, can deal with three problems by average time complexity of $O(1)$.

2.1.2 Addressing Contention and Pollution

GCLOCK [62, 78] assigns a counter to each content. The counter is increased if the content gets hit. To evict a content, the pointer of policy circularly searches

the cache until finding a content with counter of zero. The non-zero counters are decreased during the process of searching. Frequency Based-FIFO [35] divides the cache into two FIFO segments. FB-FIFO creates a protected segment in the cache for objects that are requested more than once within a short time span. The unprotected segment is specified to the contents which are not frequent. When a cache starts, the protected segment size is zero. The size of the protected segment gradually is increased by getting hits in the unprotected segments and moving the hit contents to the protected segment until the protected segment size reaches a threshold. After that, the protected segment size (consequently unprotected segment size) becomes constant.

2.1.3 Addressing Thrashing and Pollution

Unified Buffer Management (UBM) [46] automatically detects thrashing and pollution and stores the detected contents in separate partitions managed by appropriate replacement policies. The appropriate policy is selected based on detected problem (thrashing or pollution). However, UBM should address another problem of partitioning the cache. LIRS [42] and ARC [57] are the ancestors of CLOCK-Pro [41] and CAR [9] respectively. These policies separate the frequent and recent contents in different sections and adaptively change the portion of each section in the cache based on their stored additional meta-data of evicted contents. SEQ [34] is an adaptive content replacement for virtual memory management. SEQ applies Most Recently Used (MRU) policy to the long sequence of content misses with continuous addresses. For other references, SEQ performs the LRU replacement.

Early Eviction LRU (EELRU) [77] keeps meta-data for 2.5 times of the cache size. EELRU changes the eviction point of the resident contents if significant number of hits can be achieved from the meta-data of evicted contents. 2Q [76] has three queue called *A1in*, *A1out* and *Am*. The missed contents are initially placed at *A1in*. By replacing a content from *A1in*, 2Q puts the meta-data for that content in *A1out*. Only the contents getting hit in *A1in* or their meta-data

are in A_{1out} can be moved to A_m . Every content in A_m gets hit will be moved to LRU position of the A_m . Multi-Queue (MQ) [94], uses m (typically, $m=8$) LRU queues (Q_0, \dots, Q_{m-1}). The i^{th} queue contains the contents which has been referenced at least 2^i but not more than 2^{i+1} . MQ also has another queue called Q_{out} to maintain the meta-data for evicted contents. Based on the size of the Q_{out} , MQ can deal with thrashing.

2.1.4 Addressing Contention

FIFO is one the simplest cache replacement policies which does not suffer from contention. It is due to the fact that there is no action on hits. However, FIFO cannot adapt to the workload changes as good as LRU. An approximation of LRU which inherit the characteristics of LRU except contention problem is CLOCK [24]. CLOCK uses one bit per content in the cache called reference bit. The default value of the reference bit is zero when a content is written into the cache. If a content gets a hit, its reference bit is set to one. The CLOCK named is due to the circular organization of the cache. To replace a content, a CLOCK pointer passes contents until it reach a content with reference bit zero. Then, that content will be replaced. While the pointer passing the over the contents to find the victim, the reference bit of passed contents is reset to zero.

2.1.5 Addressing Pollution

LRU-K [64] keeps the times of the last K references to the cached contents and replace the content with the largest K^{th} -to-last reference. Although for simplicity, the authors recommended $K = 2$, the time complexity of LRU-K (even $K=2$) is logarithmic in the cache size. Frequency-based replacement (FBR) policy [70] divides an LRU list into three sections: new, middle, and old. In addition, every content in the cache has a counter. If a content gets a hit, the content is moved to the MRU position of the cache and its counter is increased if the content is in the middle or old section. On a cache miss, the content with the smallest counter in

the old section is replaced. However, the policy needs to rescale all the reference counts to prevent cache pollution due to the contents having big counters but no recent usage.

Segmented-LRU (S-LRU) [45] divides and LRU cache into two segments: probationary and protected. The missed contents are inserted into probationary segment and they will be promoted to the protected segment if they can get at least one hit before evicting. Although S-LRU can solve the pollution, it even intensifies the thrashing problem. That is the S-LRU decrease the minimum reuse distance that is tolerable for LRU by decreasing the probationary opportunity.

2.2 Network of Caches Perspective

2.2.1 Coordinated Hierarchical/Distributed Web Caching

Danzig et al. in [25] show that a hierarchical arrangement of several caches can decrease the amount of network bandwidth required for file transmission. In this caching scheme each cache independently decides whether to contact other caches or contact the original server. Internet Caching Protocol (ICP) is responsible to manage this contact and to check whether they have a missed request or not. By using ICP, a missed request leads to an ICN message exchange and finding a cache with the corresponding data. In Adaptive Web Caching, another hierarchical web caching scheme proposed in [58], each node has some information about the caches in its neighborhood. Caches in adaptive web caching network are formed in overlapping multicast groups. These groups have an implicit hierarchical structure. Each cache exchanges cache state message with other caches in its group. By receiving a request, a cache first checks whether the requested item can be found in its group or not. Otherwise, the cache will forward the request to another group which is more likely to have the request. Another hierarchical web caching scheme, Summary Cache [2], keeps a summary for the directory of each cache, so that the decision for request forwarding in the time of a miss can be made more precisely.

However, it imposes high overhead to the network for exchanging messages due to update in cache state. In Summary Cache, a cache only sends update messages to other caches when a threshold of its cache has been updated. Another study is affinity based collaborative web caching [92] that takes into account the affinity of caches for a missed request forwarding decision.

In addition to hierarchical web caching, distributed web caching [66] is another scheme to improve the performance of multiple caches. In this scheme each cache contacts an upper level server which has some information about each document location. Distributed web caching has shorter transmission time than hierarchical caches. On the other hand hierarchical caches have shorter connection time. Rodriguez et al. [71] made a hybrid scheme to use the advantage of both hierarchical and distributed schemes.

Neither the hierarchical nor the distributed coordinated caching schemes can be applied in ICN network of caches. It is due to the fundamental difference between hierarchical/distributed caching and ICN network of caches. The requests in the hierarchical/distributed caching are routed in such a way that increases the probability of getting hit in some nearby caches. However, ICN network of caches routes the requests based on FIB table. Requests are checked in all the caches through their paths from consumers towards the producers. In other word, ICN routing determines the caches that should be checked to find the requested item. However, hierarchical/distributed caching schemes determine the route. Another difference is that hierarchical/distributed caching is in the application level. However, the caching in ICN network of caches is implemented in the network layer.

The above discussion is also valid for the methods that are working in Content Distribution Networks (CDN). That is in CDN the route is determined in such a way that a request reaches appropriate content in a closer/more balanced server. However, in ICN finding the requested content is doing when the request is traveling in the network in its usual path. Therefore the works that have been done

for CDN cannot be applied in ICN network of caches.

2.2.2 Coordinated en-route Web Caching

Bhattacharjee et al. [11] proposed to integrate cache and router. In their proposed architecture each router caches the passing content for future uses and routing was not affected by caching. This mechanism of web caching is called en-route web caching. A more advanced coordinated scheme for en-route web caching has been proposed by Tang et al. [82]. In their en-route web caching scheme, each router caches a content in coordination with other routers that are involved in the content delivery. The authors use a dynamic programming algorithm to optimize the solution of the content placement problem. Content placement problem is the problem of putting different contents in different caches to reach a specific goal such as maximizing the traffic served by caches. An example is used to describe the proposed coordinated algorithm. In the scheme, each node maintains some information for each content such as content size and access frequency. As depicted in figure 2.1, suppose that a node A_n requests the content R which is located at node A_0 . When the request is issued by node A_n , all the nodes A_i on the path between A_n and A_0 piggybacks the corresponding information for content R . When A_0 receives the request uses the piggybacked information and computes the optimal location for caching the content R in the path. Then A_0 puts its decision together with content and sends it back to A_n . Through the path the intermediate nodes adjust their caches based on the decision. If a node is selected to cache the content, it uses a greedy heuristic algorithm to select the replacement candidates.



Figure 2.1: System model for coordinated en-route caching

All the methods in [82, 52, 53] that solve the placement problem as an optimization problem impose overhead to the routers. These methods require each

node in the network to maintain some information such as the frequency of access for each content. However, an ICN router cannot afford the overhead. It should be mentioned that based on [5] very low complexity cache management policies such as Random and FIFO can be implemented on an ICN router because the routers with caches should be able to operate in line speed.

There are less complex coordinated caching schemes. These simple schemes can be applied to ICN network of caches. Authors in [49, 16] showed that the inefficiency of hierarchical caching is due to the redundant copies of one content in different caches. That is a missed content is written to all caches from its current location (a producer or a cache) towards the consumers. So there are multiple copies from one content that leads to waste of cache space. The authors proposed meta algorithms that decrease the number of duplicated copies in hierarchical caching. There are six different meta algorithms in the literature:

- Leave Copy Everywhere(LCE)
- Prob
- Leave Copy Down(LCD)
- Move Copy Down(MCD)
- Filter
- DEMOTE

Leave Copy Everywhere (LCE) [40] refers to the usual method of uncoordinated caching. In LCE if a miss happens, the missed content will be written to all of the *intermediate caches* which are located between the location of hit for the content (a producer or a cache) and the consumer. This algorithm has the most redundancy among all of the meta algorithms.

Prob [49] is the randomized version of the LCE. Under Prob, each missed content coming to the intermediate caches will be written with a fixed probability

p and will not be written with probability $1 - p$. If $p = 1$ then Prob converts to LCE.

In Leave Copy Down (LCD) [49] a missed content only will be written to the first intermediate cache from current location of the content. LCD gradually moves contents from producer towards the consumer.

Move Copy Down (MCD) [49] is similar to LCD except that if the current location of the content is a cache, it has to evict the hit content. Four different meta algorithms are depicted in the figure 2.2.

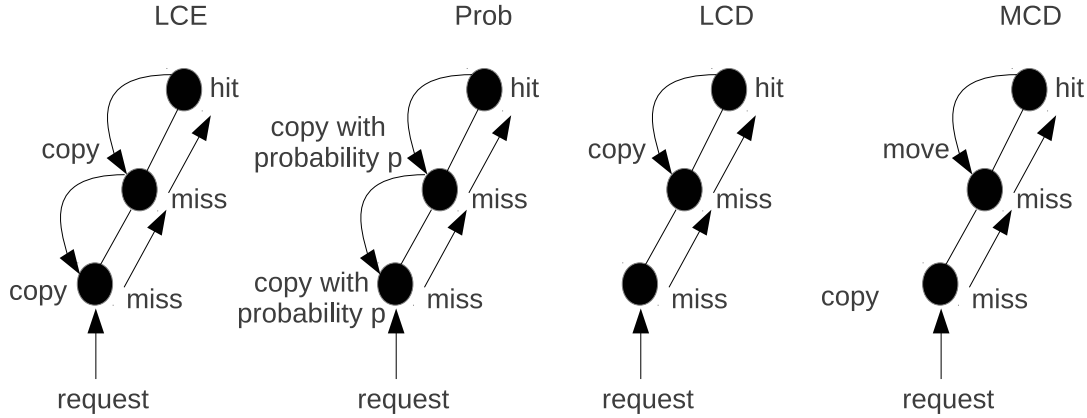


Figure 2.2: Different meta algorithms

Filter [16] is described using an example. Suppose that client k requests content i and this request is missed. Then it is forwarded until it is hit at a cache. This hit leads to writing of the content i in the intermediate cache m if the intermediate cache m satisfies the following condition: $\tau_m^{-1} < \lambda_{ki}$. τ_m is the characteristic time of the cache m under LRU replacement policy. “It is equal to the difference between the current time and a timestamp that indicates the time of last access to the document that would be replaced to make room for the caching of if LCE was to be used“ based on [16]. λ_{ki} is the frequency of requests for the content i by the client k . In this scheme, each cache is seen as low pass filter which its cutoff frequency is τ_m^{-1} . The contents that have request frequency lower than the frequency of cache filter should pass the cache without writing the contents to the cache. It is due to the fact that the probability of getting hit before being replaced is small. The

Filter meta algorithm cannot be applied in an ICN router due to its high overhead for measuring λ_{ki} .

The last meta algorithm, DEMOTE that is designed for hierarchical caching, was proposed by Wong et al. [89]. A cache which is going to evict a content passes the content to the upper level cache. The upper level cache put the content to the head of its LRU cache. On the other hand if a cache passes a content to its lower cache, the lower cache puts the content to the tail of its LRU list.

The studies that have been done in [48, 89] show that LCD has the best performance in terms of average distance to hit among all meta algorithms for hierarchical web caching. The studies are limited to the maximum of three levels of hierarchy. In the hierarchical caching the traffic moves in one direction (from the upper layer towards the lower layers). However, in ICN network of caches the traffic moves in two directions (cross traffic). To the best of our knowledge the sole investigation about meta algorithm for cross traffic is done in [73] which shows that there is no difference between LCD and LCE where there is cross traffic.

2.2.3 Coordinated Caching for ICN

In this section, the work that has been done to study the efficiency of the network of caches has been reviewed. The works are divided into three categories. First, the work that has been done to study the performance of network of caches will be reviewed in both aspects of simulation and modeling. Although some of these works are not related to the coordinated caching, their findings can give us some hints for better coordinated caching design. The second types are the works that tries to change the routing to reach better hit ratio from the caches. The last group is the work that tried to enhance the performance of network of caches by coordinated caching schemes. In each section we will describe the advantages and disadvantages of the works.

The Efficiency of ICN Network of Caches

The most complete work to study the efficiency of the network of caches has been done in [73]. Authors do an extensive simulation study for the ICN network of caches. The authors show that the most important factors for the cache performance in NoC are popularity and catalog settings. In addition, they show that the random replacement policy has the same performance as the more complex policies. In their study they consider three meta algorithms: Prob, Leave Copy Everywhere (LCE) and Leave Copy Down (LCD). The authors conclude that there is no difference between different meta algorithms. It shows that the meta algorithm that works well for hierarchical caching may not improve the performance of ICN network of caches in presence of cross traffic. Authors in [32] investigate the impact of mixing different traffic types in network of caches. Although their study is limited to uncoordinated caching, their finding is interesting. The study shows that ICN network of caches can gain more if VoD traffic is cached in access routers and other types of traffic are cached at high capacity disks in the core. A trace driven analysis of caching in ICN network of caches has been done in [84]. The authors show that caching in network of caches is beneficial. They also claim that network of caches offers more benefits than the edge caching in stub network. Arianfar et al. [5] describe a design for a content centric router. They use random autonomous caching that does not need any coordination between routers. Authors go to the detail of the functionalities and different parts of an ICN router such as ContentStore structure, insertion and deletion to ContentStore, method for lookup and caching policy. They also show that their design is practical in terms of processing throughput, memory latency, storage capacity and energy consumption. Finally they implement their design by Network Simulator and conclude that the design has significant reduction of flow completion time.

Integrating Coordination and Routing

The common property of the works in [27, 28, 85, 37] is that they announced the contents of each cache to the other caches. A cache can send their missed requests to another cache that previously announced it has the corresponding content. However, we think the small cache size compare to the total contents leads to a high rate of replacement in a cache. The high replacement rate leads to many update messages. The update messages impose a huge amount of updating overhead to the network. So we think that these methods cannot be applied in the network of caches due to their high overhead.

Fundamentally different from [27, 28, 85, 37], the authors in [75] propose to use a hash function to distribute different data packets through different routers inside an Autonomous System (AS). In addition, ASs cooperate to have different data packets. The hash value determines the router that a data packet should be cached. Based on the hash value, the request should be forward to a determined router. The determined cache may use any replacement policy. If the forwarded request gets missed in a cache, it is forwarded towards the producer. The scheme is simple to implement but the hash function role is critical and may impose high overhead because the hash function may determine a router in an AS far from the producer of a specific data packet. Therefore, if the request is missed, it should pass through several extra routers and links to reach the appropriate producer. Moreover, the corresponding data packet is also pass the extra links and routers. This may impose high overhead to the network because of useless transmissions.

Coordinated Caching Schemes

An autonomic cache management is proposed in [79]. The authors proposed distributed cache managers to decide about a content location in the network of caches. Each cache manager has a holistic network-wide view of all the cache configurations and requests patterns such as popularity. They also assumed that a cache state consistently changes with other caches. However, providing such

a wide-network view for all managers impose a huge amount of overhead to the network. In addition, gathering some required information such as popularity of different contents imposes an extra overhead to the router. So the proposed cache management scheme is not applicable in the ICN network of caches.

In [59], authors propose an age-based cooperative caching scheme for ICN. The scheme calculates the age of each packet which is supposed to be written to the cache based on two parameters: i) distance from the provider ii) popularity of the packet. The authors assume that an algorithm for calculating the popularity of different content is given by [12]. However, they do not explain how the algorithm is implemented in the proposed caching scheme. Also there is no investigation for the cost the algorithm to determine the popularity in their scheme. The proposed scheme is not applicable in network of cache similar to the previous work in [79].

In [51], authors propose an Aging Popularity-based Caching (APC) scheme. The scheme has an Interested Content Object (ICP) table to keep three additional information for each content: aging key, the latest access time and an indicator for caching state. Although authors claim that the scheme only keeps ICP only for a fraction of all content in the network, adding even a limited size table may not be affordable for an ICN router. In addition, the scheme does address the contention, thrashing and pollution problems.

CoRC [20] is a coordinated routing and caching scheme which combines the routing and caching to obtain higher hit ratio. The scheme partitions the whole content name space and assigns each partition to a dedicated node. The scheme mitigates the routing scalability to enhance caching efficiency without exchanging the control message. Although the scheme removes the filter effect problem by partitioning the name space, it increases the cost of a missed because a missed request may not be in the right direction towards the producer. Therefore, a missed request may need to be redirected to the main producer after getting a miss. Moreover, the content also should be written to a cache which may not be the reverse path of the request. This leads to communication overhead.

In [15], authors investigate the idea of not caching all packets in all routers. The study shows caching packets only on one specific router through the path from consumers towards producers can achieve better performance compare to ubiquitous caching. The specific router is the router that has the highest betweenness-centrality. However, the authors suppose that the betweenness centrality that is depend on the network topology and path information on the router can be calculated off-line. The study is the first study that reveals preventing duplicate copies of packets can offer better performance in the ICN network of cache when there exists cross traffic.

[38] proposes a probabilistic algorithm called Prob-PD based on two variables: i) the popularity ratio of a content, and ii) the distance ratio of each node from the producer. The authors define two methods for measuring the popularity: static and dynamic. However, measurement of popularity with a very large catalogue size imposes high processing and memory overhead. In addition, the evaluation is only for binary tree without cross traffic.

In a similar approach, MAGIC [68] also consider the hop distance and the popularity of the content to decide about the location of the caching node. In addition, the authors use the request packet to determine the caching node. This decreases the communication overhead affordable for ICN routers. However, the evaluation does not cover cross traffic. In addition, measuring the popularity imposes a high processing overhead to the routers.

A traffic engineering based collaborative caching has been proposed in [90]. In the proposed method each router has to measure some metric for the collaborative caching (CC) such as popularity of each packet. In addition, each router measures some metrics for the traffic engineering such as the fraction of a content that the router gets from any other router in the network. Then all of the measured information is passed to an administrative domain. The administrative domain solves an optimization problem to minimize the maximum of link utilization. The cost of information gathering of collaborative caching (CC) makes this approach inap-

plicable to the ICN network of caches. In addition, the communication overhead for exchanging of measured information between nodes and administrative domain consumes a portion of bandwidth.

In [30], authors proposed CATT architecture to deal with two fundamental problems in information centric networking: i) intelligently select one of multiple replicas distributed in the network ii) caching these contents in network. For caching part they select one point from consumer towards producer for caching a content to prevent duplicate copies. They proposed three different types of caching mechanisms: Topology aware (TP), Traffic aware (TF) and random (RD). TP caching mechanism selects the node with high degree (the number of attached links) to cache the content. TF caching mechanism selects the node with high degree of betweenness-centrality similar to [15]. Finally RD randomly selects a node. Using simulation they show that TP mechanism is the best among these caching mechanisms in terms of experienced latency by user. Interestingly they found that if more than 45% of nodes have cache, the difference between different caching mechanisms disappears. That is if more than 45% of nodes have caches a random selection of a node in the path for caching a content can get the same delay as other methods.

A collaborative caching algorithm, WAVE, is proposed In [19]. The authors used the popularity to determine the number of packets from a specific content that should be cached in the network. They use the number of requests that is served by the content producer as a metric of popularity. The number of packets that should be cached is increased exponentially when the number of served requests for a specific content is increased. The idea of using producer's knowledge is interesting. However, since there are multiple caches through the path from consumers to a producer, it is not clear which packet request should be considered as an update for popularity in the provider. If the author consider the requests for all packets, then the larger contents seems to be considered more popular than what they are. In addition, the intermediate caches filter the requests for more popular contents.

So the measurement of the popularity in the producer side is not accurate where there are multiple caches between consumers and producer.

In [67], a probabilistic in-network caching scheme is proposed. The scheme consider three different parameter to find the probability of writing a content in a cache: the total cache size in the path, the distance (hop count) from the previous location of the cache and the distance (hop count) to the consumer. The scheme has the ability of tuning its parameters such that the average times of writing a content in all of the caches in the path from the hit location towards the consumer is equal to one. The authors use traditional hierarchical topology for performance evaluation of their scheme. The scheme decreases the producer hit for 10% compare to the Leave Copy Everywhere (LCE). In addition, it decreases the number of cache-eviction in the order of magnitude compare to Leave Copy Everywhere (LCE). It is required to do simulation for the situation where there exist cross traffic since this is the real traffic situation where the network of caches is going to operate.

Another probabilistic caching scheme, LUV-Path, is proposed in [17]. The scheme uses the LUV replacement policy [7] by taking into account the distance between the routers and the producers. The LUV has two phase evaluation and normalization. LUV evaluates a data packet to predict its likelihood of being re-referenced based on the past references. Then, LUV normalizes the likelihood value by the cost of the object per unit size. The LUV-Path uses LUV to give higher probability to the popular data packets to be cached by the routers close to the consumers. In addition, LUV-Path gives higher probability to the unpopular data packets to be cached by the routers far from the consumers to. However, the scheme suffers from filtering effect.

Table 2.1: Related work summary - single cache perspective

Ref	Contention	Thrashing	Pollution	Time Com.	Mem. Ov.
RND	No	No	No	$O(1)$	0
CAR [9]	No	No	No	$O(1)$	$2C$
CLOCK-Pro [41]	No	No	No	$O(1)$	C
LRFU [50]	No	No	No	$O(\log n)$	n
GCLOCK [62]	No	Yes	No	$O(1)$	0
FB-FIFO [35]	No	Yes	No	$O(1)$	0
UBM [46]	Yes	Yes	No	$O(1)$	0
LIRS [42]	Yes	No	No	$O(1)$	$3C$
ARC [57]	Yes	No	No	$O(1)$	C
SEQ [34]	Yes	No	No	$O(1)$	0
EELRU [77]	Yes	No	No	$O(1)$	$2.5 \times C$
2Q [76]	Yes	No	No	$O(1)$	C
static [83]	No	No	No	$O(\log n)$	N
MQ [94]	Yes	No	No	$O(1)$	C
LRU-K [64]	Yes	Yes	No	$O(\log n)$	0
FBR [70]	Yes	Yes	No	$O(1)$	0
S-LRU [45]	Yes	Yes	No	$O(1)$	0
CLOCK[24]	No	Yes	Yes	$O(1)$	0
FIFO	No	Yes	Yes	$O(1)$	0
LRU	Yes	Yes	Yes	$O(1)$	0

2.3 Summary

2.3.1 Single Cache Perspective

In this part, we summarize the related work of cache management algorithms from a single cache perspective. The comparison is depicted in Table 2.1 based on five challenges described in Section 1.2.1. It should be mentioned that the hit ratio which is a very important metric for evaluating cache management algorithms is missing in the table. This is due to the fact that the hit ratio drastically varies with cache size and workload changes. Therefore, we compare our policy with the state of the art cache management algorithms, ARC and LIRS, as well as the popular cache management policies such as LRU and FIFO in Section 5.4.

As it can be seen from the table, although RND policy can overcome all of the challenges, it suffers from the low cache hit ratio compared to LRU, ARC

and LIRS. RND is the base replacement policy in one of our cache management policies, CAP. Therefore, CAP can tackle all the challenges and at the same time obtain high hit ratio close to ARC and LIRS.

2.3.2 Network of Caches Perspective

In this part, we summarize the coordinated caching schemes for the network of caches in Table 2.3.2 based on the ten different challenges described in Section 1.2. It should be mentioned that **DoR** is the abbreviation for Depend on Replacement. DoR is used to describe that the challenge is depend on the cache replacement policy used for each cache. For example, the work [58] can overcome the first four challenges listed in the table if RND is used in as the cache replacement policy. In contrast, [58] cannot deal with the first four challenges if the LRU is used as the replacement policy in every cache. Moreover **NI** is the abbreviation for Not Investigated. That is, the challenge has not investigated yet.

The contention happens in a router which has several ports accessing a common cache. In addition, thrashing is a common problem for current routers with the small cache size (maximum of 10GB[5]) and huge amount of data in the Internet. Moreover, pollution is also a common problem for in-network and ICN network of caches. This is due to the fact that high amount of traffic is one-timer [55, 86].

Method	Ref	Hit Con.	Miss Con.	Thrash.	Poll.	Comp.	Mem. O.	Filter	Com. O.	Red.	Cr. Tr.
Hierar. Distr.	[25]	DoR	DoR	DoR	DoR	DoR	DoR	Yes	Low	High	No
	[58]	DoR	DoR	DoR	DoR	DoR	High	No	High	Med	No
	[2]	DoR	DoR	DoR	DoR	DoR	High	No	High	Med	No
	[92]	DoR	DoR	DoR	DoR	DoR	High	No	High	Med	No
	[66]	DoR	DoR	DoR	DoR	DoR	High	No	High	Med	No
	[71]	DoR	DoR	DoR	DoR	DoR	High	No	High	Med	No
En-Route Coordination	[11]	Yes	Yes	Yes	Yes	$O(1)$	Low	Yes	Low	Med	NI
	[82]	Yes	Yes	Yes	Yes	$O(1)$	Low	No	High	Med	NI
	[52]	Yes	Yes	Yes	Yes	$O(\log C)$	Low	No	High	Low	NI
	[53]	Yes	Yes	Yes	Yes	$O(\log C)$	Low	No	High	Low	NI
	[89]	Yes	Yes	Yes	Yes	$O(1)$	Low	Yes	Low	Low	Poor
	LCE[40]	Yes	Yes	Yes	Yes	$O(1)$	Low	Yes	Low	High	Poor
	Prob[49]	Yes	Yes	Yes	Yes	$O(1)$	Low	Yes	Low	Med	Poor
	LCD[49]	Yes	Yes	Yes	Yes	$O(1)$	Low	Yes	Low	Low	Poor
	MCD[49]	Yes	Yes	Yes	Yes	$O(1)$	Low	Yes	Low	Low	Poor
	Filter[16]	Yes	Yes	Yes	Yes	$O(1)$	High	No	High	Low	NI

Method	Ref	Hit Con.	Miss Con.	Thrash.	Poll.	Comp.	Mem. O.	Filter	Com. O.	Red.	Cr. Tr.
ICN Coordinated	[68]	Yes	Yes	No	No	$O(1)$	High	No	Low	Low	NI
	[20]	Yes	Yes	Yes	No	$O(1)$	High	No	High	Low	Good
	[27]	Yes	Yes	Yes	Yes	$O(1)$	High	No	High	Low	Poor
	[28]	Yes	Yes	Yes	Yes	$O(1)$	High	No	High	Low	Poor
	[38]	No	Yes	No	No	$O(1)$	High	No	Low	Low	NI
	[85]	Yes	Yes	Yes	Yes	$O(1)$	High	No	High	Low	Poor
	[51]	Yes	Yes	Yes	Yes	$O(1)$	High	Yes	Low	Low	NI
	[37]	Yes	Yes	Yes	Yes	$O(1)$	High	No	High	Low	Poor
	[79]	Yes	Yes	Yes	Yes	$O(1)$	High	No	High	Low	Poor
	[15]	Yes	Yes	Yes	Yes	$O(1)$	High	No	High	Low	Poor
	[19]	Yes	Yes	Yes	Yes	$O(1)$	Low	Yes	Low	Low	Poor
	[67]	Yes	Yes	Yes	Yes	$O(1)$	Low	Yes	Low	Low	NI
	[17]	Yes	Yes	Yes	Yes	No	$O(1)$	High	Yes	Low	Low

As it is depicted in Table 2.3.2, the related works of hierarchical/distributed web caching have either high communication overhead or high redundancy. Among related work of en-route web caching, [16, 52, 53, 82] have high overhead to implement. They are not simple enough to be implemented in an ICN router too. Also LCE[40] leads to a high redundancy in network of caches. Prob, LCD, MCD proposed in [49] and DEMOTE proposed in [89] do not have good performance in network of caches with cross traffic.

Among related work for ICN, [27, 28, 85, 37] propose schemes for changing routing algorithm to get more from caching which is not applicable due to the high rate of content replacement. [79, 15] have high overhead. Also [19] has poor performance in network of caches with cross traffic. There are two works that are compatible with ICN and they have low overhead and redundancy[11, 67]. However, they are suffering from the first four problems: hit-contention, miss-contention, thrashing and pollution. Moreover, their performance is not investigated under a network of caches with cross traffic.

Chapter 3

Two-State Cache Management Policy

In this chapter, we start to build the basics for our first approach of proposing the lightweight coordinated schemes for ICN network of caches. First, we explain our two-state policy in terms of managing a single cache. We evaluate the two-state cache hit ratio versus LRU, RND and FIFO (referred by replacement policies for the rest of the thesis), the applicable replacement policies for an ICN router [5], using both synthetic and real workloads. Next, we introduce the main advantage of the two-state policy in terms of obtaining a high overall hit ratio in the ICN network of caches and compare our policy versus LRU, RND and FIFO in terms of their effects on the overall cache hit ratio. Finally, we show that large RTTs degrade the hit ratio of two-state in a standalone cache because large RTTs affect the adaptability of two-state policy. To deal with this issue, we introduce a mechanism (reservation) that enables a cache to adapt to the traffic pattern changes even when the RTT is large. We conclude the chapter with a summary.

Before moving to the description, we need to emphasize on two characteristics of the cache size in an ICN network of caches. These characteristics should be taken into account in designing a caching policy or coordinated caching scheme. First, the ratio of cache size compared to the catalog size, i.e. the total number of

contents, is very small (in the order of 10^{-5}) [74]. Second, there is not a hierarchy of caches in which each level has a very larger cache size compared to the caches of which it receives the requests. That is, the caches (located at the core routers) that receive the requests through other caches have the cache size in the same order as the cache at the edge routers that receives the requests directly from consumers. Therefore, we consider these characteristics for our experiments and presents some of the results in this chapter and our complementary results in the Appendix A, B, C and D.

3.1 Two-State Policy in Single Cache

The main objective of proposing two-state policy is to deal with filter effect because the filter effect degrades the overall hit ratio in ICN network of caches. We propose to freeze the cache for a predefined amount of time. This freezing helps the subsequent caches to obtain a high hit ratio. We explain how two-state operates in this section.

3.1.1 Description

The main idea of two-state policy is depicted by the state transition diagram of Figure 3.1, where a cache is operated at either an *updating state* or a *frozen state*. After a cache transits to the frozen state, its contents remain *frozen* for a predefined amount of time called *updating period*. While the cache is in frozen state, no new content is cached and no existing content is evicted, before the cache transits back to the updating state. Under the two-state policy, two design

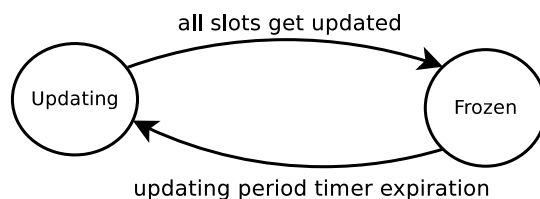


Figure 3.1: State diagram of two-state cache management

parameters need to be determined: 1) the updating period, τ , 2) the triggering condition for transition to the frozen state.

Regarding the τ value (updating period), as two-state policy does not change the cached contents in the frozen state, the cache should be refreshed when contents become unpopular because of changes in routing paths, network topologies, user demand, and etc. To adapt to the various network changes in practice, the updating period should be set shorter than the times between these exogenous changes. For example, the popularity of VoD contents does not change much within a day [21, 14] and most (around 2/3) of Internet paths do not change for days [65, 54]. Hence, suitable lengths of the updating period can be in the order of hours in practice.

Regarding the triggering condition, a cache considers all of its slots as *outdated* after entering the updating state and updates all of its slots before transition to frozen state (triggering condition). A cache updates an *outdated* slot by either i) getting hit for its content, or ii) writing a missed content. Consequently, a cache captures the first C distinct requested contents and then returns to the frozen state. In terms of implementation, the cache should differentiate between outdated and updated slots. To do so, a cache uses one extra bit for each slot called *update bit* (Ub). The Ub of one (zero) indicates an updated (outdated) slot. In addition to Ub , we use a variable called *updatedSlots*, representing the number of updated slots that is increased by one whenever the cache updates a slot. Therefore, a cache transits to its frozen state when the $updatedSlots = C$ (triggering condition).

The two-state policy does not suffer from the thrashing problem because the cache gets frozen after capturing the first C distinct requested contents. Moreover, the contention only happens when the cache is in the updating state and the contention happens in the slot level. However, the two-state suffers from one-timer contents (pollution) because among the first C distinct contents captured in the updating state, there may be some one-timer contents that are not replaced

while the cache is in the frozen state. This decreases the hit ratio of two-state in presence of the one-timer. We will deal with this problem by proposing the n -state policy in Chapter 4. In the next two subsections, we show the cache hit ratio of the two-state policy versus FIFO, RND and LRU (applicable in ICN) for both synthetic and real workloads respectively.

3.1.2 Synthetic Workload Evaluation

In this section, we evaluate the hit ratio of two-state policy from a standalone cache perspective. First, we prove that the two-state policy has the same hit ratio as LRU under the Independent Reference Model (IRM) assumption by using a simple mathematical model. Later, we compare the hit ratio of two-state policy with FIFO, RND and LRU (applicable in an ICN router) using simulation under IRM assumption.

Model

Consider a set $F = \{1, 2, \dots, n\}$ of n different contents, out of which c contents can be stored in a set $S = \{1, 2, \dots, c\}$ of c cache slots. Under IRM assumption, the i^{th} most popular content is independently requested with probability q_i that is the popularity of the i^{th} content and we have $q_1 \geq q_2 \geq \dots \geq q_n$. In addition, suppose a cache is managed by a *two-state policy*. After entering the updating state, the policy places the first requested content, $\sigma_1 \in F$, in the first slot. Then, the policy places the second requested content, $\sigma_2 \in F - \{\sigma_1\}$, in the second slot and this process continues. Therefore, the cache goes to the frozen state after filling the c slots with the first c distinct requested contents in order. Let us assume the $\vec{\sigma} = (\sigma_1, \sigma_2, \dots, \sigma_c)$ represents the state of the cache in the frozen state where $\sigma_i \in F$ is located at the i^{th} slot.

Lemma 1. *The probability of finding a cache managed by the above described*

two-state policy in the frozen state of $\vec{\sigma}$, $P(\vec{\sigma})$, is obtained through

$$P(\vec{\sigma}) = \prod_{i=1}^c \frac{q_{\sigma_i}}{1 - \sum_{j=1}^{i-1} q_{\sigma_j}} \quad (3.1)$$

Proof. Let p_j^i denote the probability that the i^{th} distinct requested content is content $j \in F$ given that the first $i-1$ distinct requested contents are $\sigma_1, \sigma_2, \dots, \sigma_{i-1}$. For $i=1$, $p_{\sigma_1}^1$ is simply the probability that the first requested content is the content σ_1 . Therefore, we have

$$p_{\sigma_1}^1 = q_{\sigma_1} \quad (3.2)$$

,and

$$p_{\sigma_i}^i = \frac{q_{\sigma_i}}{1 - \sum_{j=1}^{i-1} q_{\sigma_j}} \quad (3.3)$$

The explanation is that given the first $i-1$ distinct requested contents are $\sigma_1, \sigma_2, \dots, \sigma_{i-1}$, the remaining contents compete to occupy the i^{th} cache slot. We exclude the popularity of the already requested contents and normalize the popularity of the remaining contents to one. The probability of finding a cache in a frozen state $\vec{\sigma}$ can be calculated by the following expression.

$$P(\vec{\sigma}) = \prod_{i=1}^c p_{\sigma_i}^i = \prod_{i=1}^c \frac{q_{\sigma_i}}{1 - \sum_{j=1}^{i-1} q_{\sigma_j}} \quad (3.4)$$

□

Theorem 1. *Under IRM assumption, $P(\vec{\sigma}) = \pi_{LRU}(\vec{\sigma})$ where $\pi_{LRU}(\vec{\sigma})$ is the steady state probability of finding an LRU cache in the state of $\vec{\sigma}$.*

Proof. Based on [81], $\pi_{LRU}(\vec{\sigma})$ can be calculated by

$$\pi_{LRU}(\vec{\sigma}) = \prod_{i=1}^c \frac{q_{\sigma_i}}{1 - \sum_{j=1}^{i-1} q_{\sigma_j}} \quad (3.5)$$

that is the same with $P(\vec{\sigma})$ found in Lemma1. □

Simulation

Setting: We set an experiment with topology depicted in Figure 3.2 to compare

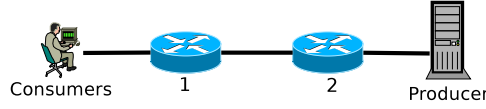


Figure 3.2: Simple topology for filter effect experiment

the first cache hit ratio achieved by the two-state policy and replacement policies (FIFO, RND and LRU). In the experiment, the routers have equal cache size and there are N equally sized contents located at the producer. In addition, the consumers generate content requests based on Zipf(α , N) distribution where α is the slope of the distribution. We do the simulation with 2×10^7 requests and updating period is 5×10^4 requests. In addition, the RTT in this experiment is zero and have the experiments with $RTT > 0$ in Section 3.3.

Findings and Discussion: Figure 3.3 shows that the first cache hit ratio for

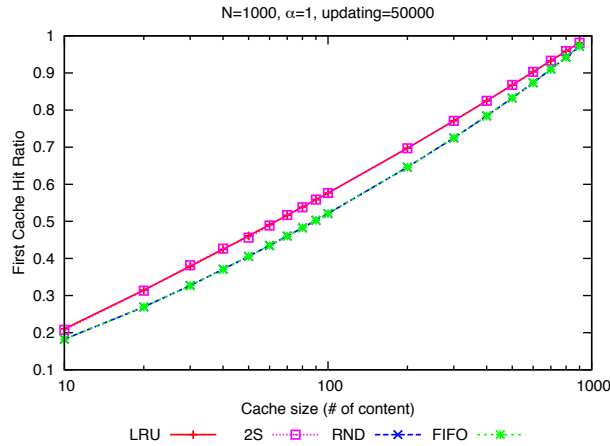
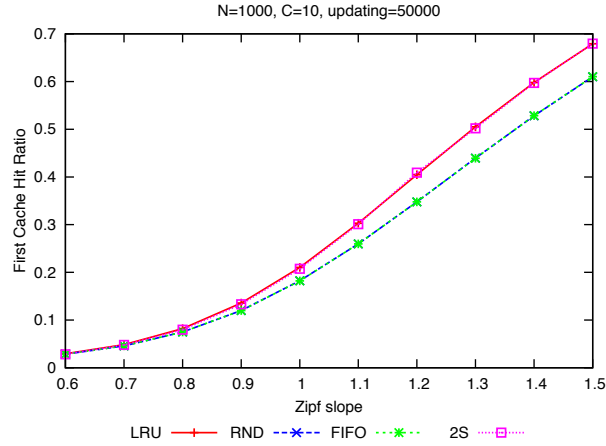


Figure 3.3: The first cache hit ratio with different policies versus cache size

LRU and the two-state policy are the same under the IRM assumption for different cache sizes. In addition, Figure 3.4 shows that the first cache hit ratio is also the same with LRU for a wide range of α (Zipf slope) with $C = 10$.

Figure 3.4: The first cache hit ratio with different policies versus popularity (α)

3.1.3 Real Workload Evaluation

Although using synthetic workload gives us the opportunity of changing the characteristic of the workload such as α in Zipf distribution, we evaluate our findings through trace-based simulation too. The specification of the traces is presented in Table 3.1.

Setting: We use eight traces of IRCache [39], used in recent studies [91], from

Trace	Total req.	Total contents	1-timer req.	1-timer contents	Max. hit ratio
bo2	448875	264460	50.7%	86.0%	67.7%
ny	843925	545348	55.2%	85.4%	35.3%
pa	487179	229287	47.0%	83.6%	43.7%
rtp	6162823	2991227	43.0%	88.6%	51.4%
sd	2923802	1720736	51.0%	86.6%	41.1%
sj	289879	711434	33.9%	83.2%	59.2%
sv	354382	975442	27.9%	76.7%	63.6%
uc	533406	1074618	42.4%	85.5%	50.3%

Table 3.1: The specifications of the traces

eight different proxy caches for the period of 2007/01/09-2007/01/10. The maximum hit ratio is obtained by an infinite cache size.

Findings and Discussion: Figure 3.5 shows that the two-state hit ratio in the first cache is less than the replacement policies. This contradicts with our

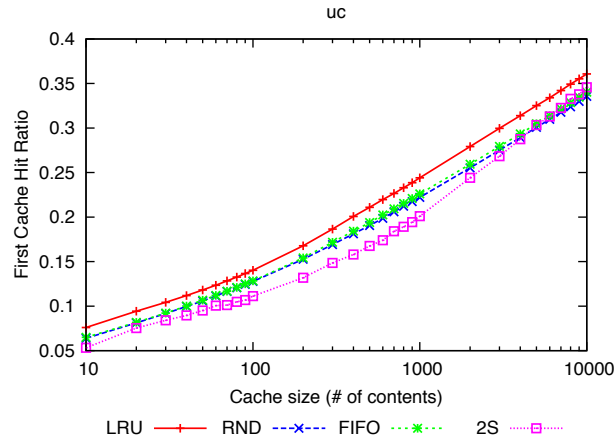


Figure 3.5: The first cache hit ratio with different policies versus cache size with *uc* trace

finding under the IRM assumption because real traces have a high percentage of one-timer contents (pollution problem) as depicted in Table 3.1. The high percentage of one-timer contents causes the first cache to go to the frozen state with a number of one-timer contents that cannot be evicted for updating period amount of time. This degrades the hit ratio of the first cache compared to the LRU, RND and FIFO that are able to evict the one-timer contents faster than two-state. Although the difference between LRU and two-state is small for trace

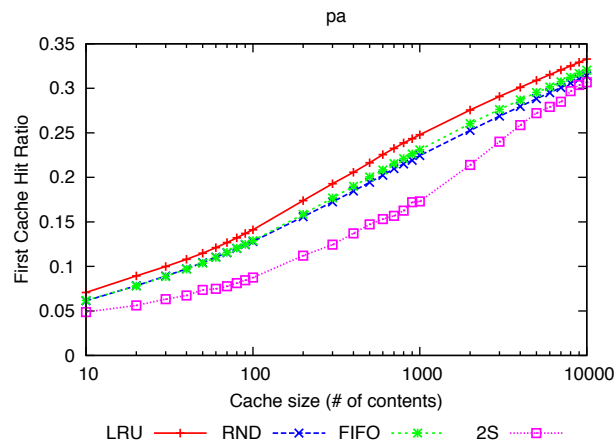


Figure 3.6: The first cache hit ratio with different policies versus cache size with *pa* trace

uc in Figure 3.5, Figure 3.6 shows that the situation can be worse for other traces. Therefore, we introduce a generalized version of the two-state policy called N-state

policy, in Section 4.1, that solves the pollution problem for the real workloads. So far, we have evaluated the hit ratio of two-state policy for a single cache. In the next section, we explain and discuss how the two-state policy improves the overall network hit ratio compared to the replacement policies.

3.2 Two-State Benefits for Network of Caches

Our objective in this section is to show that using the two-state policy at the edge routers provides better opportunity for core routers to obtain high hit ratios compared to the situation where the edge routers are managed by LRU, RND and FIFO. First, we explain the reason behind providing the better opportunity and then evaluate our description through the simulation of both synthetic and real workloads.

3.2.1 Description

The performance of a standalone cache is influenced mainly by its management policy. However, in an ICN network of caches, a cache performance is influenced not only by its management policy but also by the interactions with other caches. For example, one of the interactions in a network of caches is the filter effect [88] that is caused by replacement policies and lowers the overall hit ratio by serving the requests that generate cache-hits and forwarding the requests that generate cache-misses. Hence, there is little chance for core routers to achieve high hit ratios because their incoming requests are filtered by the edge routers. However, our *two-state policy* mitigates the filter effect by introducing a new filtering type.

The filter effect lowers the overall hit ratio because it affects the locality of reference used by replacement policies to obtain high hit ratios. The locality of reference means that “a content just requested has a high probability of being referenced in the near future” [44]. In other words, the locality of reference determines the *potential* of achieving a high hit ratio. The stronger the locality of

reference is, the more the potential of achieving a high hit ratio exists. However, the locality of reference is weakened by the filter effect of the replacement policy but the *two-state policy* allows the missed requests to be efficiently served by other caches. We should emphasize that there are multiple interpretations of locality of reference in the literature. We use the interpretation in [31] that defines two kinds of temporal locality (locality of reference): popularity and correlation. We use the term locality of reference to cover both popularity and correlation in this thesis.

Although both two-state and replacement policies serve some requests (hits) and forward some requests (misses), they have different types of filter effect on the pattern of requests. Despite the replacement policy that serves a *fraction* of the requests (with strong locality of reference) of **all** contents, the two-state policy serves *all* the requests for C (**cache size**) number of contents. To make the difference clear, we use an example. Suppose that in Figure 3.2 (simple topology for filter effect experiment) both routers use replacement policy. In this situation, router1 can count on the locality of reference of the receiving requests. However, the locality of reference is weakened in router2 because the requests are affected by router1. That is, if router2 receives a request for a specific content, router2 cannot assume that it will receive another request for that specific content with a high probability in near future. Otherwise, router1 should miss two requests with strong locality of reference. This contradicts with the functionality of replacement policy in router1. However, by using the *two-state policy* in router1, router2 still is able to count on the locality of reference because router1 either serves all of the requests of one specific content or forwards all of the requests for that content. Therefore, if router2 receives a request for a specific content, router2 can assume that it will receive another request for that specific content with a high probability in near future. In the next subsection, we evaluate the above discussion regarding different types of filter effect.

3.2.2 Synthetic Workload Evaluation

In this section, we firstly use stack distance metric, used in the literature to quantify the locality of reference, to show that the missed requests from a cache with our two-state has a stronger locality of references compared to the missed request of the caches managed by LRU, RND and FIFO. Then, we show that this leads to a higher hit ratio obtained by serving the missed requests from a two-state policy than LRU, RND and FIFO.

Metrics Explanation

The stack distance is widely used in the literature to characterize the locality of reference [56]. The stack distance of the j^{th} request ($j = 2, 3, \dots$) for content i is defined as the number of *distinct* contents requested between the $j - 1^{\text{th}}$ and j^{th} requests for content i (undefined stack distance considered for the first request of content i). For example, let 4, 5, 1, 3, 2, 7, 2, 3, 1, 6 be a stream of requests for contents 1 to 7. The stack distance of the second request for content 1 is three because there are three distinct contents (2, 3, 7) requested between the first and the second requests of content 1. The stack distance represents the strength of locality of reference. The smaller the stack distances of the content requests are, the stronger the locality of reference for the requests of that content is. Using the stack distance, we define three metrics to characterize the locality of reference: the minimum, maximum and average stack distances.

The minimum (maximum) stack distance is defined as the smallest (largest) stack distance seen in a stream of requests. The minimum (maximum) stack distance in combination with cache size affects the hit ratio. To explain this impact, let us assume that a stream of requests with minimum stack distance of 10 enters to a cache with less than 10 slots. In this situation, FIFO or LRU obtains hit ratio of zero because they evict a content before being re-referenced. In contrast, RND obtains a hit ratio greater than zero because the cache does not evict all of the contents before being re-referenced. On the other hand, the

maximum stack distance determines the minimum cache size that is required to obtain the hit ratio of 1 (excluding cold misses).

The average stack distance (SD_{avg}), our last metric based on stack distance, is defined as

$$SD_{avg} = \frac{\sum_{i=0}^{n-1} SD(i) \times i}{\sum_{i=0}^{n-1} SD(i)} \quad (3.6)$$

where n is the total number of contents in a stream and $SD(i)$ is the number of occurrences of stack distance i in the stream. Although the SD_{avg} is more representative for locality of reference compared to minimum and maximum stack distances, it is not sufficient to characterize the locality of reference by itself. For example, two streams with similar SD_{avg} may lead to totally different hit ratios because of different minimum stack distances. We use these three metrics to explain how the replacement policies and the *two-state policy* differ in changing the locality of reference.

Setting

We use the same topology from the previous set of experiments depicted in Figure 3.2. We measure the minimum, maximum and average stack distances of the missed requests from the first cache while the first cache managed by FIFO, RND, LRU and two-state. In addition, we measure the second cache hit ratio while it is managed by FIFO, RND and LRU. Similar to previous set of experiments, the routers have equal cache size and there are N equally sized contents located at the producer. The content requests are generated based on Zipf(α, N) distribution where α is the slope of the Zipf distribution. We do the simulation with 2×10^7 requests.

Findings and Discussion (Stack Distances vs Cache Sizes)

As it can be seen from Figure 3.7, the minimum stack distance of the missed requests from a cache managed by *two-state policy* is zero because *two-state policy* filters requests based on content not based on locality of reference. That is, *two-*

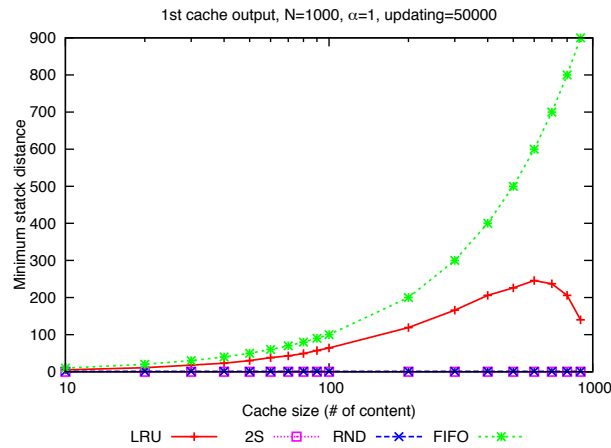


Figure 3.7: Minimum stack distances of the missed requests from the first cache managed by three replacement policies and two-state policy

state policy only filters the requests for a specific set of contents and forwards the rest (even two consecutive requests for the same content). Therefore, *two-state policy* gives the opportunity to other caches to use the remaining locality of references. Close to the minimum stack distance of *two-state policy*, RND has the minimum stack distance of 1. Totally different from two-state and RND, the minimum stack distance of FIFO increases linearly with the cache size and is equal to the cache size (C) because FIFO evicts a content when there is exactly C number of misses after the time that the content entered the cache. Similar to FIFO, the minimum stack distance of LRU increases by increasing the cache size but only up to a specific point and the increment is less than FIFO because LRU evicts a content by receiving C distinct requests rather than C distinct misses. That is, for LRU, some of the requests contributing in the eviction of a content are hits (filtered by the cache). Therefore, the minimum stack distance for LRU can happen with a smaller number of misses compared to FIFO. In addition, the fraction of hits that push a content towards LRU position (contributing to evict the content) is increased by the increment of the cache size and after a certain point (600 in this example) overcomes the fraction of missed requests. Therefore, after the specific point, the minimum stack distance is decreased by increasing the cache size.

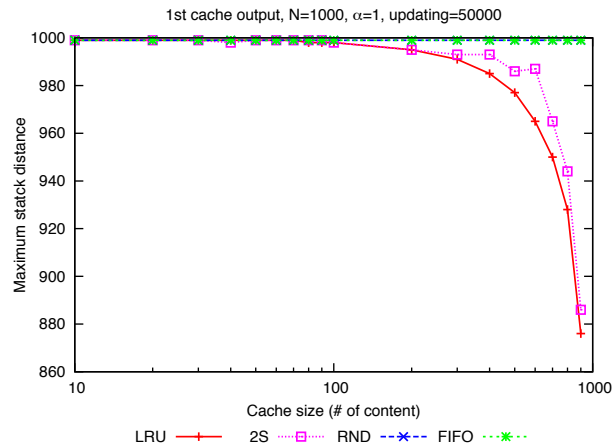


Figure 3.8: Maximum stack distances of the missed requests from the first cache managed by three replacement policies and two-state policy

The maximum stack distances of missed requests, depicted in Figure 3.8, determines the minimum cache size for the second router to reach the hit ratio of 1. As depicted in Figure 3.8, the maximum stack distances of RND and FIFO do not decrease by increasing the cache size because of the evicting mechanism of these policies. That is, they evict a content independent from its number of hits. For example, FIFO even evicts the most popular content by getting C distinct misses after writing the most popular content. In addition, RND probably evicts the most popular content by getting even one miss after writing it. Therefore, FIFO and RND evict all the contents in the system. Consequently, it is possible to get $N - 1$ (N is the catalogue size, total number of contents in the system) misses between two consecutive misses. In contrast to FIFO and RND, LRU decreases the maximum stack distance. To make the description of the LRU maximum stack distance curve easier, we assume that the maximum stack distance happens for two misses of the least popular content. The decrement in LRU maximum stack distance is due to considering content hits for eviction. That is, LRU keeps a popular content in the cache for a long time by moving the popular content to the MRU position whenever it gets hit. Therefore, the most popular content may not get missed between two misses of the least popular content (the misses lead to maximum stack distance). Moreover, the larger the cache size, the more the

number of non-missed contents between two misses of the least popular content. Similarly, *two-state policy* decreases the maximum stack distance by the cache size while the cache is in the frozen state. That is, for cache size of 10, the missed requests have the maximum stack distance of 989 because 10 contents are excluded from the missed stream. However, the plot shows almost the same maximum distance for *two-state* and LRU because we measure the maximum even while the cache is in the updating state and operates similar to the replacement policies. If we only measure maximum stack distance when the cache is in the frozen state, the maximum stack distance is decreased by the cache size for *two-state policy*.

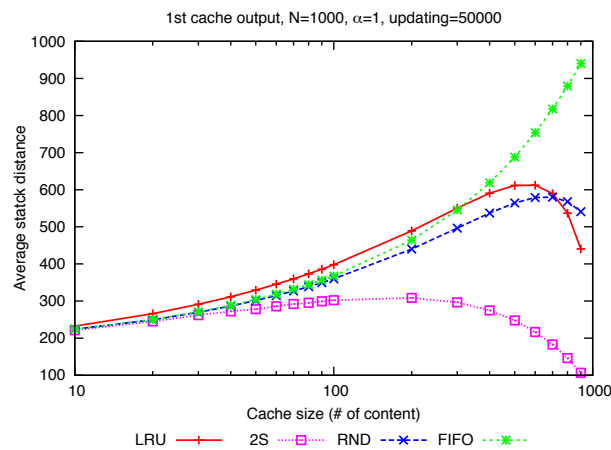


Figure 3.9: Average stack distances of the missed requests from the first cache managed by three replacement policies and two-state policy

Regarding the average stack distance, it has the least increment when the *two-state policy* manages the first cache as depicted in Figure 3.9. In addition, the average stack distance of two-state is decreased after cache size 200 and reaches below its initial value, 190, for cache sizes larger than 600. However, other policies increase the average stack distance in a way that it never reaches below the initial value. Especially, FIFO increases the average stack distance linearly because FIFO increases the minimum stack distance and keeps the maximum stack distance constant. However, LRU increases the minimum stack distance slower than FIFO and starts to decrease the maximum stack distance for cache sizes larger than 300 as depicted in Figure 3.7 and 3.8. Consequently, the average stack distance of LRU

becomes smaller than FIFO for cache sizes greater than 300. On the other hand, RND and LRU increase the average stack distance up to $C = 600$ because the first cache hit ratio reaches around 90% that makes a number of popular contents almost resident in the cache and excluded from the missed stream.

To conclude, we show that by using the two-state policy at the first router, the second router has a higher opportunity to obtain a high hit ratio. We show this finding for 1000 contents of which requests generated based on Zipf law with Zipf slope of 1. In the next section, we show that our findings are also valid for the same number of contents with a fixed cache size while the Zipf slope, α , varies.

Findings and Discussion (Stack Distances vs Zipf Slope)

To recall the experiment settings, we use same topology and there is the same number of contents, 1000, on the producer. However, we select the cache size of 10 due to the small fraction of cache size to the catalogue size in ICN network and change the popularity by changing the α .

As it can be seen from Figure 3.10, the two-state policy has the smallest minimum stack distance of zero and RND has the minimum stack distance of one

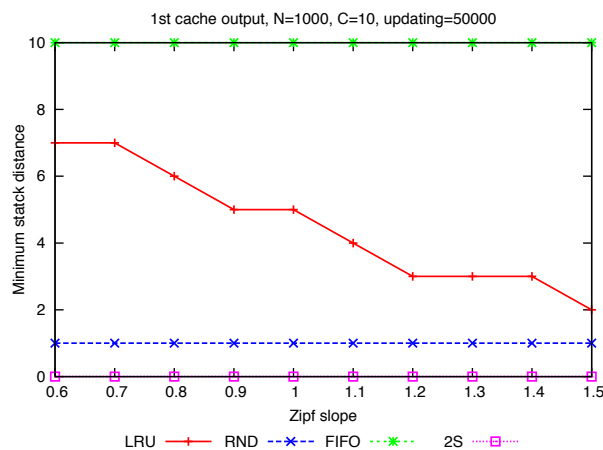


Figure 3.10: Minimum stack distances of the missed requests from the first cache versus different α (Zipf slope)

(reason explained above). On the other hand, FIFO has the largest minimum stack distance that is equal to the cache size for different α . Finally, the LRU minimum

stack distance decreases with increasing the α because the misses, forwarded to the next cache, are decreased by increasing the α . Therefore, the minimum stack distance is decreased.

The maximum stack distance is constant for different α and for all policies as depicted in Figure 3.11. This shows that increasing the α increases the number of hits but not enough to make some of the contents residents in the cache with size 10.

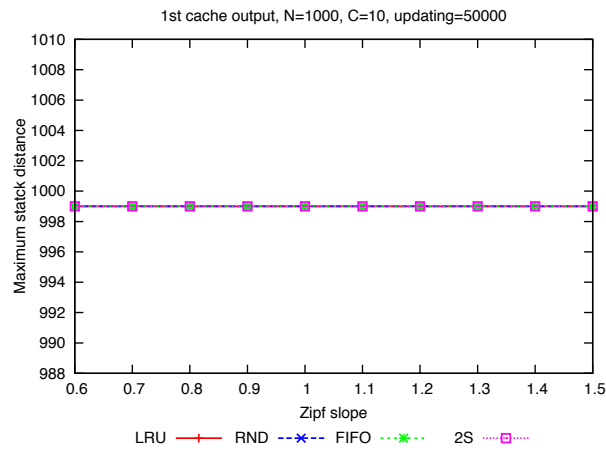


Figure 3.11: Maximum stack distances of the missed requests from the first cache versus different α (Zipf slope)

As depicted in Figure 3.12, the average stack distance is decreased by increasing the α for all policies. Our two-state policy has the smallest average stack distance for $\alpha < 1.1$ but RND and FIFO have the smallest average stack distance after this point. This may be interpreted that RND and FIFO under large α provide a better situation for the subsequent caches to obtain a high hit ratio. However, we show in the next section that the second cache obtains the highest hit ratio while the first cache managed by the two-state even for $\alpha > 1$. To determine the reason for this phenomenon, we use the Cumulative Distribution Function (CDF) of the stack distance and the fact that the LRU hit ratio for a stream of requests can be obtained by using its CDF. That is, the LRU hit ratio of a cache with C slots can be obtained by $CDF_{SD}(C - 1)$ where CDF_{SD} is the CDF function of the stack distance because an LRU is able to capture the requests with stack distance less

than C . In addition, [10] proves that the LRU obtains higher hit ratio than FIFO

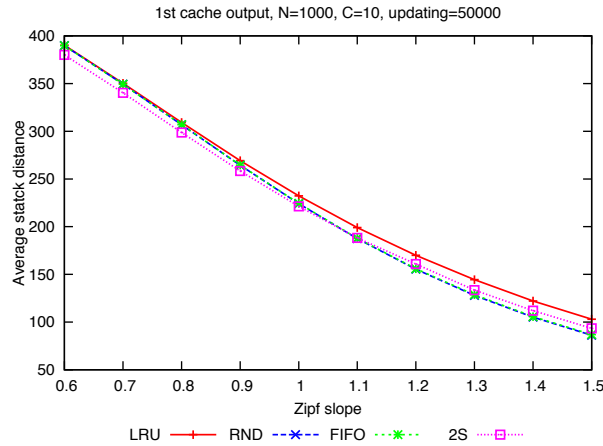


Figure 3.12: Average stack distances of the missed requests from the first cache versus different α (Zipf slope)

and RND under IRM. Therefore, the CDF of the stack distance can determine the potential of obtaining hit ratio.

Depicted in Figure 3.13, the two-state CDF with $\alpha = 1.2$ has the largest value for small cache sizes up to the cache size of 30 (three times of the first cache size of 10). Moreover, Figure 3.14, representing the CDF with $\alpha = 1.5$, shows that

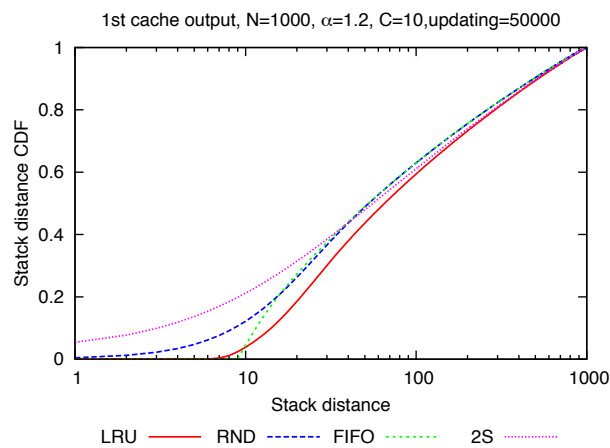


Figure 3.13: The CDF plot of the stack distance from the missed requests with $\alpha = 1.2$

the two-state policy has the largest CDF value for the small cache sizes up to cache size of 20. The comparison of these figures implies that increasing the α decreases the cache size that the two-state still has the largest CDF value but

we should consider two points regarding this finding. First, as described before

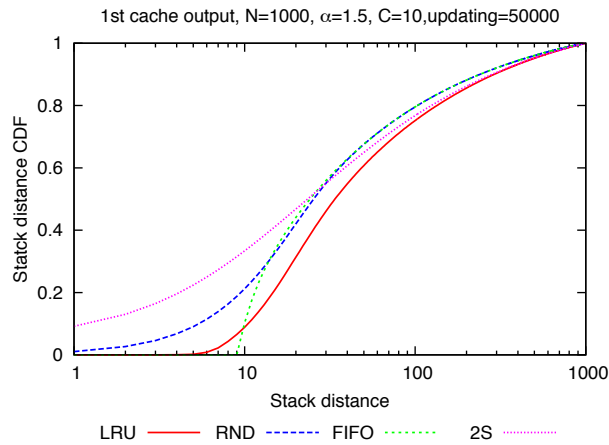


Figure 3.14: The CDF plot of the stack distance from the missed requests with $\alpha = 1.5$

in ICN network of caches, the different routers have either equal or in the same order cache sizes. Second, the hit ratio of the first cache is around 70% for $\alpha = 1.5$ (shown in Section 3.1.2). In such situation, there is not much motivation for using the network of caches. Therefore, the interval of cache sizes that two-state has the highest hit ratio is suitable for ICN network of caches.

We can conclude that, the locality of reference of the missed requests from the *two-state policy* has the highest potential to be efficiently served by another cache most of the time. Specifically, we show that this is true for small cache sizes relative to the catalogue size that is the case for ICN network of caches. Consequently, the second cache (managed by LRU, RND and FIFO) has the highest hit ratio when the first cache is managed by the *two-state policy*. This is shown in the below discussions.

Findings and Discussion (Second Hit Ratio vs Cache Size)

Figure 3.15 shows that RND in the second cache obtains its highest hit ratio while the first cache managed by two-state. The figure follows the trend of average stack distance depicted in Figure 3.9. For example, RND obtains a higher hit ratio with FIFO than LRU in the first cache up to the cache size of 300 but obtains higher

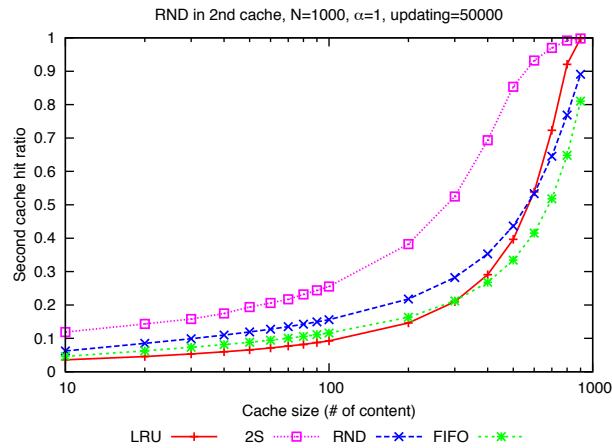


Figure 3.15: The hit ratio of the second cache that is managed by RND. The first cache is managed by RND, LRU, FIFO and two-state policy

hit ratio with LRU for cache sizes greater than 300. There is a similar trend for average stack distances in Figure 3.9 where the average stack distance of LRU starts to become less than FIFO at $C = 300$. The last but not the least, RND

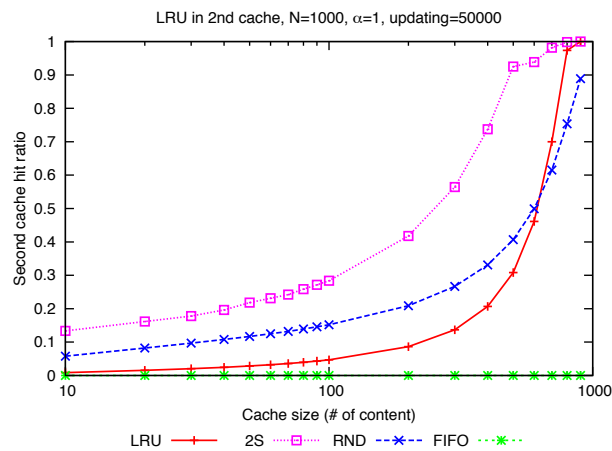


Figure 3.16: The hit ratio of the second cache that is managed by LRU. The first cache is managed by RND, LRU, FIFO and two-state policy

obtains a non-zero hit ratio with all four combinations.

The hit ratio of the second cache managed with LRU, depicted in Figure 3.16, is zero while the first cache managed by FIFO because of minimum stack distance. As depicted in Figure 3.7, the minimum stack distance of the missed requests from FIFO is equal to the cache size. On the other hand, the second cache has the same

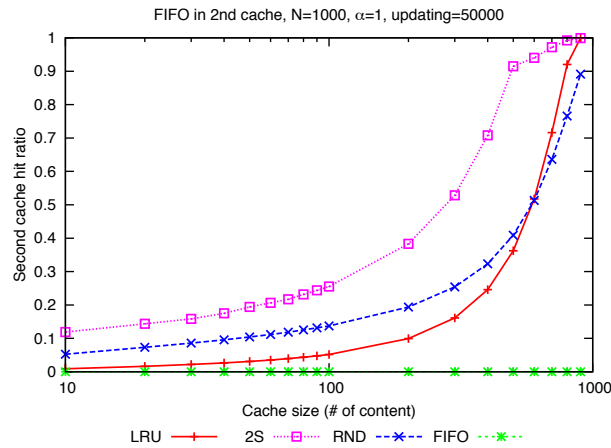


Figure 3.17: The hit ratio of the second cache that is managed by FIFO. The first cache is managed by RND, LRU, FIFO and two-state policy

size as the first cache and this leads to the hit ratio of zero at the second cache. The same reasoning causes that the hit ratio of the second cache managed by FIFO becomes zero while the first cache managed by FIFO in Figure 3.17.

Findings and Discussion (Second Hit Ratio vs Zipf Slope)

Figure 3.18, 3.19 and 3.20 show that second cache obtains the highest hit ratio while the first cache is managed by two-state policy for different α . Similar to Figure 3.15, 3.16 and 3.17, the hit ratio of second cache managed by LRU or FIFO is zero while the first cache managed by FIFO as depicted in 3.19 and 3.20 (described before).

This section concludes that under IRM assumption using two-state at the edge router provides better opportunity for the core router to obtain a high hit ratio compared to the situation that edge routers managed by LRU, RND and FIFO. In the next section, we are going to investigate if this property is valid when the workload is based on real traces or not.

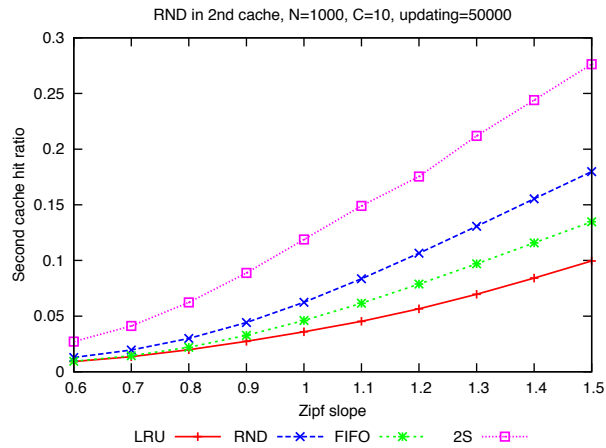


Figure 3.18: The hit ratio of the second cache that is managed by RND. The first cache is managed by RND, LRU, FIFO and two-state policy

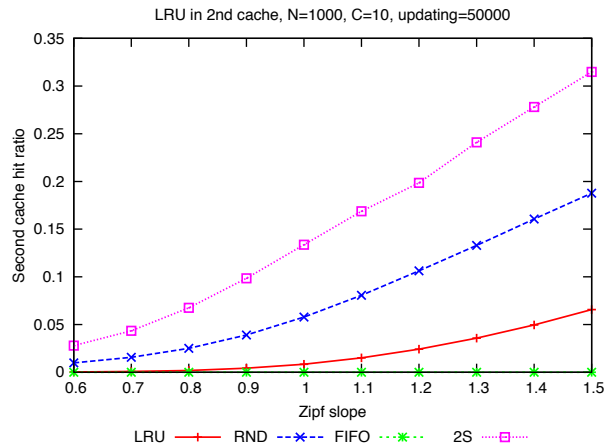


Figure 3.19: The hit ratio of the second cache that is managed by LRU. The first cache is managed by RND, LRU, FIFO and two-state policy

3.2.3 Real Workload Evaluation

In this section, we repeat the previous set of experiments using real trace-based simulation to show that using two-state at the edge router provides better opportunity for the core router to obtain a high hit ratio compared to the situation that edge routers managed by LRU, RND and FIFO.

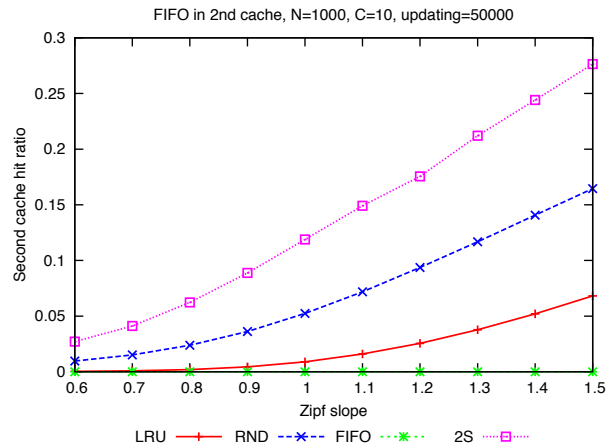


Figure 3.20: The hit ratio of the second cache that is managed by FIFO. The first cache is managed by RND, LRU, FIFO and two-state policy

Setting

We use the same topology and traces used in Section 3.1.3. The trace description can be found in Table 3.1.

Findings and Discussion (Stack Distances vs Cache Size)

Figure 3.21 shows that the minimum stack distance linearly increases for LRU and

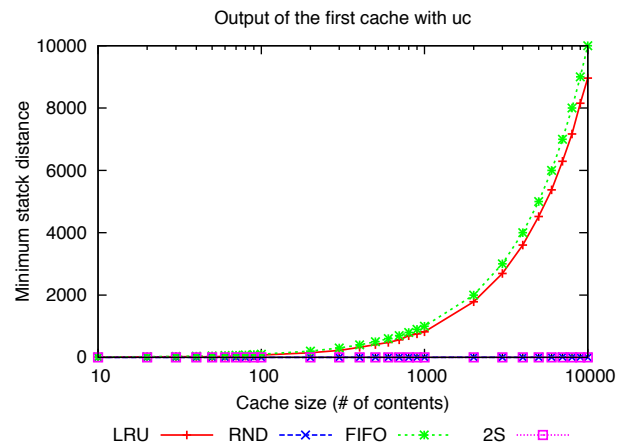


Figure 3.21: The minimum stack distances of the missed requests from the first cache with trace uc

FIFO by increasing the cache size. However, it is constant for RND and two-state for *uc* trace. The reason is because of the way that these policies are working and

described in Section 3.2.2.

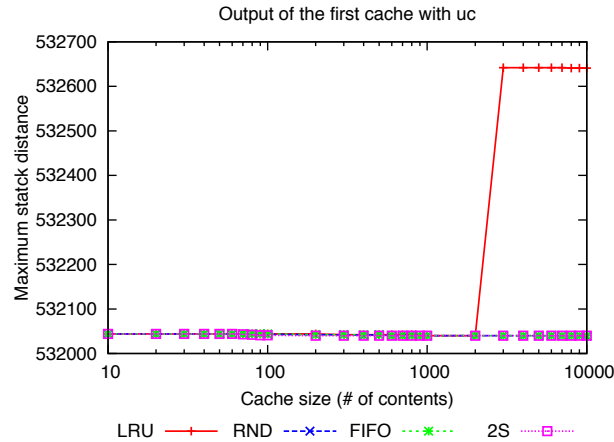


Figure 3.22: The maximum stack distances of the missed requests from the first cache with trace *uc*

Figure 3.22 shows that the maximum stack distance of *uc* trace negligibly decreases by increasing the cache size (same trend of IRM) for all policies except LRU. The negligible decrease is due to the effect of one-timer contents that weakens the effect of increasing the cache sizes. For LRU, the maximum stack distance negligibly decreases up to cache size 2000 and there is a jump at 3000. After 3000, the maximum stack distance negligibly decreases. LRU jumps at 3000 because the maximum happens between two specific consecutive missed requests and for $C \geq 3000$ one of these requests gets hit in the cache and the maximum happens between other request and a further request. We only see this jump in *uc* and *pa* traces. We should mention that the effect of increasing the cache size on maximum stack distance is very negligible because of one timer. For example, the largest decrement in the maximum stack distance that is four happens by increasing the cache size from 10 to 10000. Therefore, we can conclude that the maximum stack distance is almost unchanged for real traces because one-timer contents prevent decreasing the maximum stack distance by increasing the cache size.

Finally, Figure 3.23 shows that the average stack distance of the missed requests has the smallest value for two-state policy up to cache sizes of 5000. For cache sizes greater than 5000, the average stack distance of FIFO and RND have

the smallest value. We present the results for other traces that have the same trends in Appendix B.

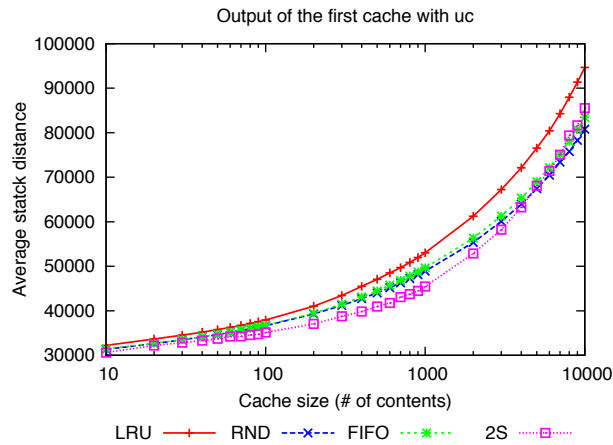


Figure 3.23: The average stack distances of the missed requests from the first cache with trace *uc*

Findings and Discussion (Second Hit Ratio vs Cache Size)

Figure 3.24, 3.25 and 3.26 show that the second cache (managed by RND, LRU FIFO respectively) obtains the highest hit ratio for real trace *uc* while the first cache managed by the two-state policy. The figures show that the hit ratio of the second cache starts to decrease after increasing the cache size up to 1000 because the updating period of the two-state policy is set to 5000 requests. Therefore, the two-state policy waits for receiving 5000 requests after entering into the frozen state and then returns to the updating state. This causes the second cache hit ratio decreases for $C > 1000$ because it does not provide enough time (requests) for the second cache to obtain a high hit ratio. Therefore, by increasing the updating period the trend of increasing the hit ratio of the second cache continues. It should be mentioned that the hit ratio of the first cache with size of 1000 is almost 50% of the maximum achievable hit ratio by infinite cache size. In such situations, the requirement of using network of caches may be doubtful.

So far, we have shown that our two-state policy provides the opportunity for subsequent caches to obtain a high hit ratio but suffers from one-timer contents.

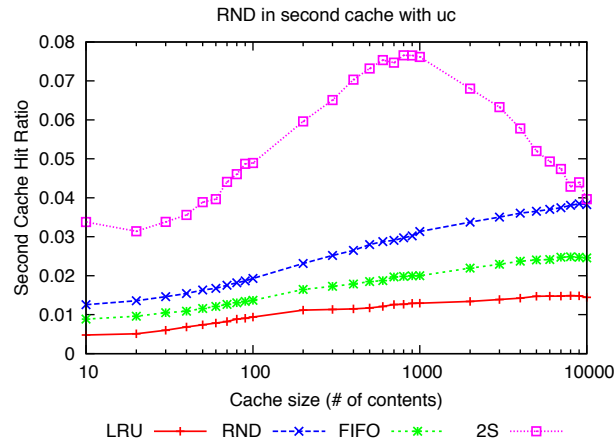


Figure 3.24: The hit ratio of the second cache that is managed by RND. The first cache is managed by RND, LRU, FIFO and two-state policy

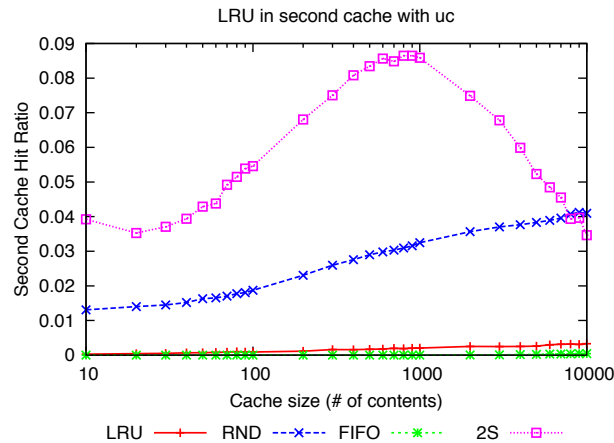


Figure 3.25: The hit ratio of the second cache that is managed by LRU. The first cache is managed by RND, LRU, FIFO and two-state policy

In the next chapter, we present our solution for this issue. In addition, the performance of two-state policy may be affected in terms of hit ratio by the network RTT. In the next subsection, we discuss this situation and explain our proposed mechanism, reservation, to deal with this RTT issue.

3.3 Popularity Changes and Reservation

The content popularity varies in Internet through time when the unpopular contents become popular and vice versa. We can provide the opportunity for two-state

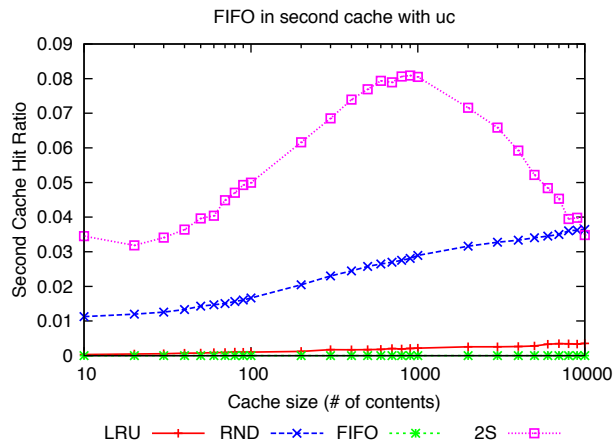


Figure 3.26: The hit ratio of the second cache that is managed by FIFO. The first cache is managed by RND, LRU, FIFO and two-state policy

policy to adapt to the popularity changes by setting the updating period smaller than the average period of popularity changes. In addition, the two-state policy is able to capture the new popular contents by capturing the first C distinct requested contents in the updating state. However, the implementation of the two-state policy so far cannot guarantee to capture the first C distinct requested contents where the $RTT > 0$. This weakens the ability of capturing the new popular contents and consequently decreases the hit ratio by increasing the RTT . In this section, we explain how increasing the RTT affects the ability of capturing the new popular contents and propose a simple mechanism, *reservation*, to deal with this situation.

3.3.1 Motivation

The time between missing a request and receiving its corresponding content is the RTT between the cache and the content producer as depicted in Figure 3.27a. With $RTT > 0$, a cache in the updating state can categorize its receiving missed contents into: i) missed contents get requested in updating state ii) missed contents previously get requested in frozen state. To provide the opportunity of a lightweight coordination, explained in Section 4.2, the two-state policy only writes the missed contents requested in the updating state. Therefore, there is

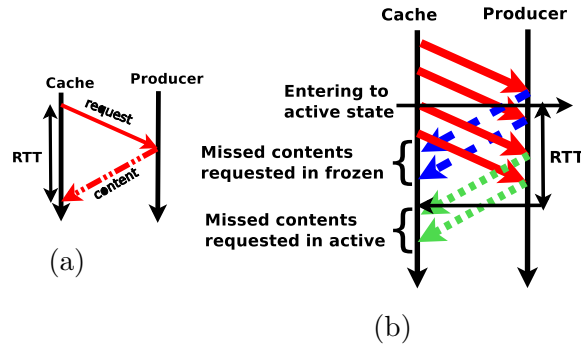


Figure 3.27: RTT between sending a missed request and receiving its content

at least one RTT between the time that a cache enters the updating state until the cache receives the first missed content as depicted in Figure 3.27b. During this period (RTT), it is probable that the contents in all cache slots get hit. This leads to updating all slots and transiting to the frozen state with the previous contents. Consequently, the two-state policy cannot capture new popular contents. In addition, increasing the RTT increases the chance of having this situation.

To show the effect of the RTT on the adaptability of two-state policy, we consider the catalog size of 1000 and Zipf slope of 1, while requests arrive based on a Poisson process with rate 10^4 requests/sec. For every random amount of time X , exponentially distributed with mean 50 seconds (rate of $\lambda_c = 0.02$), we change the popularity of contents such that 1) each content will have an equal probability to be more or less popular, and 2) the change in its popularity rank (1 to the catalog size 1000) is determined by a geometric random variable with mean 20 (success probability of $p_s = 0.05$). The p_s determines the intensity of popularity changes. For example, $p_s = 1$ leads to the situation that all of the ranks remain unchanged. However, decreasing the p_s enlarges the difference between current and new ranks. Therefore, we can obtain harsh popularity changes by setting p_s close to zero. We focus on the performance of a standalone cache and repeat the experiment with 10 runs, each lasts for 2×10^5 seconds.

As depicted in Figure 3.28, the two-state hit ratio decreases by increasing the RTT because the cache goes to the frozen state before capturing the new popular

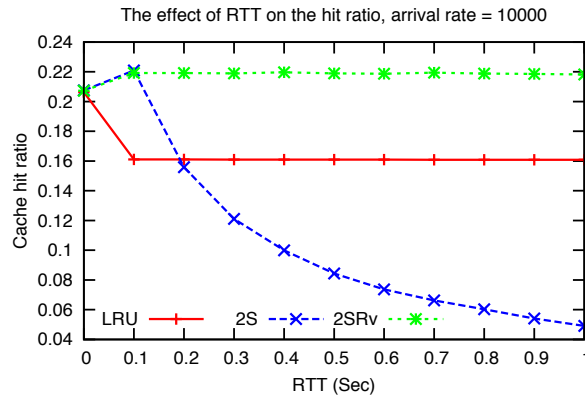


Figure 3.28: The hit ratio of the cache with arrival rate of 10000 requests per second, updating period of 10 sec, the average popularity change period of 50 sec ($\lambda_c = 0.02$) and the probability of popularity changes, p_s , of 0.05 - $C=10$, $N=1000$, $\alpha = 1$

contents. On the other hand, LRU hit ratio decreases by increasing the RTT from zero to 0.1 second but it remains constant after $RTT = 0.1$. The shape of the LRU hit ratio curve can be interpreted through the relation of RTT and characteristic time [16, 48]. The characteristic time is defined as the maximum inter-arrival time between two consecutive requests for a content that leads the second request to a hit [48]. In Figure 3.29, we represent the characteristic time of a content with T ,

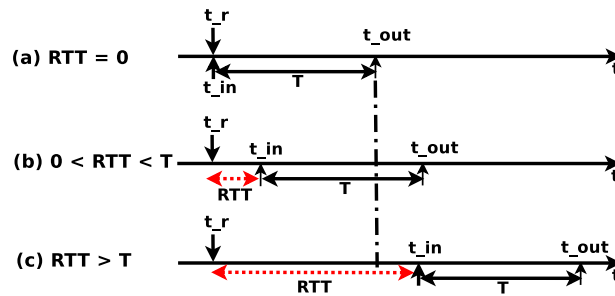


Figure 3.29: Different combinations of RTT and characteristic time that plays role in decreasing the hit ratio of LRU

the request time with t_r , the incoming time of a content with t_{in} and the outgoing time of a content in condition that the content has not been requested from its t_{in} with t_{out} ; $T = t_{out} - t_{in}$. For $RTT = 0$, the first request after a miss will get hit if it arrives at $t \in [t_{in}, t_{out}]$ and miss if it arrives at $t > t_{out}$. Having $RTT > 0$ causes the first request after a miss to get missed if it arrives at $t \in [t_r, t_r + RTT]$ because the missed content has not reached the cache. We call this kind of misses as RTT

misses because these misses are hit with $RTT = 0$. Increasing the RTT increases the number of RTT misses until RTT becomes equal to the characteristic time (T). For $RTT > T$, the number of RTT misses is equal to the number of misses with $RTT = T$. This is due to the fact that if the first request after a miss arrives to the cache at $t > t_{in} + T$, it leads to a miss. Finally, we should mention that the characteristic time is approximately the same for all contents with different popularity [48]. Therefore, after a certain RTT (T), increasing the RTT does not decrease the LRU hit ratio.

3.3.2 Reservation

A two-state with reservation policy guarantees capturing the *first* C distinct requested contents after entering the updating state even for $RTT > 0$. To do so, among the first C distinct requests arriving after entering the updating state, those lead to hit update the corresponding slots. On the other hand, the policy reserves cache slots for those that get missed from their forwarding time until the time of receiving the corresponding missed contents. Therefore, independently from RTT , the two-state with reservation always fills the cache with the *first* C distinct requested contents in the updating state. Consequently, the policy is able to capture the new popular contents even for large $RTTs$ as depicted in Figure 3.28

The reservation implementation needs two mechanisms in the updating state for i) reserving a slot ii) distinguishing between the incoming contents that get missed *before* and *after* entering the updating state. We implement the reservation by using a variable called *resSlots*, representing the number of reserved slots. A cache reserves a slot through increasing *resSlots* by one in condition that $resSlots + updatedSlots < C$. The condition prevents the cache from over reserving. In addition, a cache updates an outdated slot with $Pb = 0$ by getting hit for its content only in condition that $resSlots + updatedSlots < C$. The unsatisfied condition ($resSlots + updatedSlots = C$) at the hit time means that the cache needs the outdated slots to write the missed contents that are on the way. In addition to

reserving a slot, we need the second mechanism, distinguishing between the incoming contents that get missed *before* and *after* entering the updating state, because the two-state with reservation only writes the missed contents that are requested in the updating state. We describe the implementation of the mechanism and the coordinating reasons behind it in Chapter 4. The high level description is that a cache puts extra information in the header of forwarded requests after entering updating state. Through coordination and the extra information, the incoming missed contents that were forwarded **in** the updating state are distinguishable from others. Finally, it should be mentioned that Pending Interest Table (PIT) in NDN [40] or its equivalent in other ICN proposals handles the case that multiple misses happen for the same content in the updating state because PIT only lets the first missed request for a content to be forwarded towards the producer.

3.4 Summary

This section describes a new cache management policy, the two-state policy, that achieves two objectives: i) obtains a high cache hit ratio and, ii) lets other caches to obtain a high hit ratio by serving the missed requests from the cache managed by two-state. In terms of the first objective, we prove that under IRM assumption the two-state policy obtains the same hit ratio as LRU. However, trace-based simulation shows that the two-state policy obtains less hit ratio for a standalone cache than other policies because of the one-timer contents. In terms of the second objective, we show how the two-state policy can address the filtering problem by introducing a new type of filtering. To distinguish between the filtering effect of the replacement policy and the two-state policy, we use the minimum, maximum and average stack distances to explain how the two-state policy manages a cache such that the missed requests can be effectively used by other caches to obtain high hit ratios for both synthetic and real workloads. Finally, we introduce a mechanism called reservation that enables the two-state policy to adapt to the

traffic pattern changes even for large RTT.

So far, we have proposed the two-state with reservation, the base for our first coordinated scheme, which is able to address the contention and thrashing problems with low time complexity and memory overhead (one bit for each slot). However, the limitation of two-state policy is its low hit ratio in the presence of one-timer contents as we showed in evaluation with trace-based evaluation. To deal with one-timer contents and improve the standalone cache hit ratio, we will propose a generalized version of two-state that is the base for our coordinated scheme in Chapter 4. Our first approach to tackle the challenges listed in Chapter 1 will finish at the end of Chapter 4.

Chapter 4

Coordinated Caching Scheme

In Chapter 3, we introduced our two-state policy that provides better opportunity for subsequent caches to obtain a high hit ratio compared to replacement policies but suffers from one-timer contents. To deal with this issue in this chapter, we extend our two-state with reservation policy to n -state with reservation (summarized by two-state and n -state in the rest of the thesis). The n -state policy obtains higher hit ratio compared to two-state by removing the one-timer contents and capturing the popular contents. We start by explaining the implementation of the n -state policy for a single cache. Then, our experiments, using both synthetic and real workloads, show that increasing n (number of states) from two to three considerably improves the hit ratio but the improvement is negligible for increasing n where $n > 3$. After discussing about n -state, we complete our first approach for proposing the lightweight coordinated schemes for ICN by explaining our coordinated scheme integrated with n -state policy. Moreover, we discuss about two important properties of our scheme: managing the redundancy and caching the popular contents close to the consumers. We present the evaluation of our schemes in Chapter 6. We conclude the chapter with a summary.

4.1 N-State Policy with Reservation

As we explained in Section 3.1.3, the two-state policy suffers from one-timer contents (pollution). This leads to the situation that many of the cached contents in the frozen state are one-timer contents that decrease the hit ratio. Moreover, both prior work studying web caching workload [55] and video sharing workload [60] showed that up to around 50% of the data are one-timers. Our objective of introducing the n-state policy is to solve the one-timer problem of two-state to increase the standalone cache hit ratio.

4.1.1 Main Idea and Implementation

As explained in Section 3.1.1, all of the slots in a two-state cache are considered as outdated after the cache enters the updating state. Each slot gets updated if i) a hit happens to its content ii) a missed content for which the cache reserved a slot is written to a slot. By getting updated, a slot Pb is set to one and the cache goes to the frozen state when all of its slots get updated once. The n-state also has the same logic and a cache goes to the frozen state if all of its slots get updated $n - 1$ times. However, in an n-state cache, a slot gets updated if i) a hit happens for its content ii) a missed content is written to the slot **that has not gotten updated so far**. Moreover, the reservation is used only when the cache is in updating state (state zero). That is, writing a missed content only updates a slot if this is the first update since the cache enters updating state. Therefore, any slot that goes to frozen gets at least one hit and one-timer contents cannot reach frozen state for $n \geq 3$. To keep the track of each slot, we use a variable per slot called slot state to count the number of times that a slot gets updated after a cache enters updating state. The slot with the minimum state is the place that a cache writes a missed content and changes the slot state to $Max(1, slotState)$ where the $slotState$ is the current slot state. On the other hand, if a slot gets hit, the cache increases the state value by one except for the slots reached state $n - 1$.

Moreover, a slot reached $n - 1$ is not replaced by a missed content. Therefore, the cache does not write any missed content when all of its slots reach state $n - 1$.

The cache state diagram of n -state policy is depicted in Figure 4.1. Based on the definition of the slot state, we define the whole cache state as the minimum state of its slots. For example, a cache in the state i has at least one slot in the state i and all of its slots has the state greater than or equal to i . Therefore, a whole cache may be in one of the n different states from zero (updating state) to $n - 1$ (frozen state). The n -state uses a timer, *updating timer*, for the whole cache to adapt to traffic pattern changes similar to two-state. By the expiration of the updating timer, the cache transits to the state-0 (updating state) and resets the state of all slots to zero. When all of its slots get updated at least once (cache

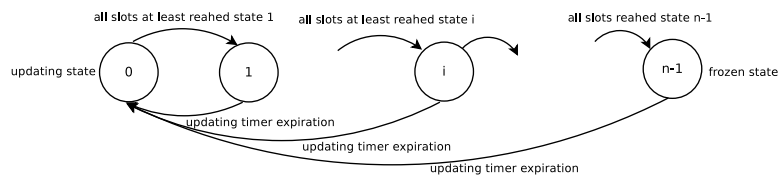


Figure 4.1: The cache state diagram in n -state policy

enters state one), the cache restarts the updating timer.

So far, we have explained the n -state mechanism that prevents the one-timer contents from reaching the frozen state. In the next two subsections, we evaluate the effect of n , number of states, on standalone and overall cache hit ratio under synthetic and real workloads.

4.1.2 State Number Effect on Standalone Hit Ratio

In this section, we investigate the effect of number of states (n) on the hit ratio of a standalone cache.

Synthetic Workload

Setting: For all experiments in this section, we use the same experiment topology used in Chapter 3 where there are two caches between a group of consumers and a

producer. The requests are generated for 1000 contents based on Zipf distribution with slope of one and the experiments are repeated for three small cache sizes of 10, 50 and 100.

Findings and Discussion (Standalone Hit Ratio vs n): Figure 4.2 and all of the figures in Appendix C show that the improvement of cache hit ratio by increasing the number of states from two to three is considerable. However, the improvement by increasing the state number from three to four, four to five and five to six is almost zero for all synthetic workloads. In Section 4.1.4, we will show that the cache hit ratio for three-state policy is pretty close to LFU that is the optimal replacement policy under IRM assumption [23]. Although, the results are limited to the IRM assumption, they indicate that three-state policy is able to capture the popular contents much better than two-state, RND, LRU and FIFO.

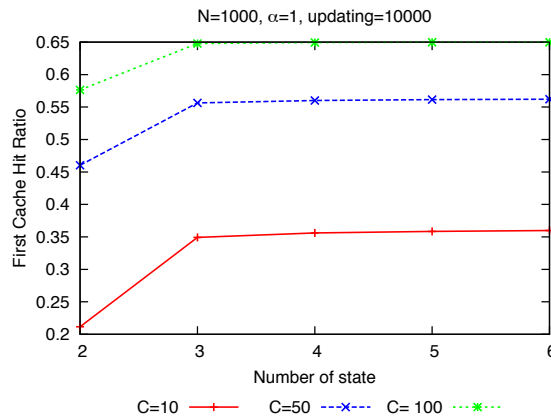


Figure 4.2: The first cache hit ratio with different policies versus cache size

Trace-Based Workload

Setting: The requests are generated based on *bo2* and *sd* traces and the experiments are repeated for three small cache sizes of 10, 100 and 1000. We select different updating periods 25000 and 50000 to show that the results are valid under different updating periods.

Findings and Discussion (Standalone Hit Ratio vs n): Figure 4.3 and 4.4 show the hit ratio of the first cache versus increasing the number of states

for two traces described in Section 3.1.3. These figures and the figures depicted in Appendix D for other six traces show that the hit ratio improvement is only

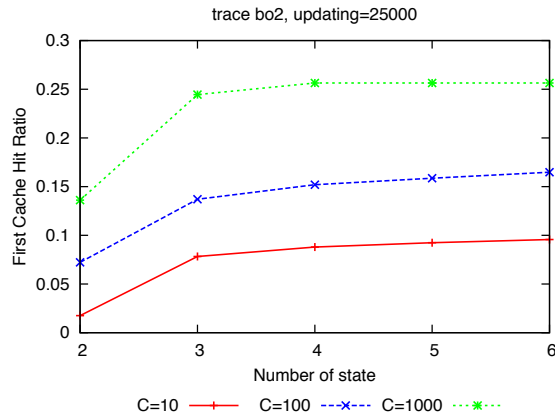


Figure 4.3: The first cache hit ratio with n -state versus number of states by bo2 trace

considerable for increasing n (number of states) from two to three. So far, we have found that $n = 3$ improves the two-state policy hit ratio. In the next subsection, we investigate the effect of n on the overall hit ratio and locality of references.

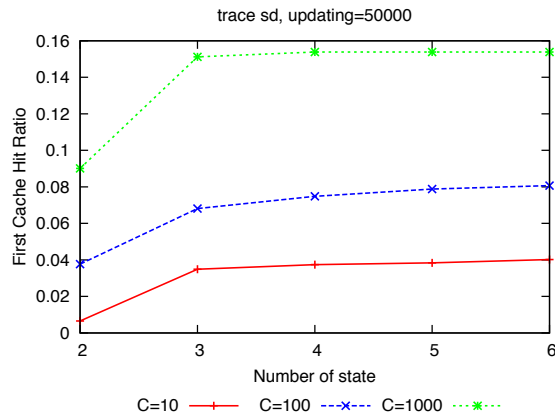


Figure 4.4: The first cache hit ratio with n -state versus number of states by sd trace

4.1.3 State Number Effect on Overall Hit Ratio

In this section, we investigate about the effect of number of states (n) on the overall cache hit ratio by measuring the hit ratio of the second cache that receives the missed requests from the first cache.

Synthetic Workload

Findings and Discussion (Overall Hit Ratio vs n): Figure 4.5, 4.6 and 4.7 respectively show the hit ratio of the second cache managed by LRU, RND and FIFO while the first cache is managed by the n -state policy. The main decrease

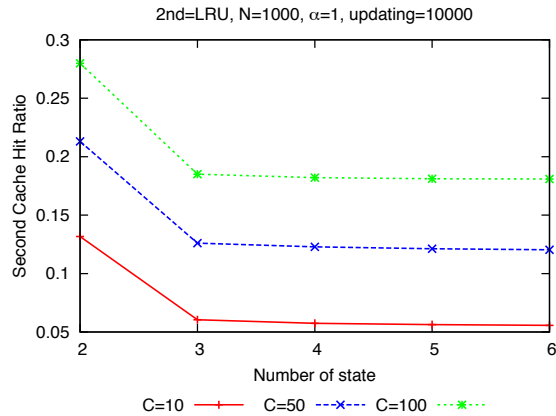


Figure 4.5: The second cache hit ratio with LRU versus number of states with $N = 1000$, $\alpha = 1$

happens when n increases from two to three because of the increase at the first cache hit ratio as shown in Figure 4.2. That is, the first cache with three-state obtains a higher hit ratio and leave less potential for the second cache to obtain a high hit ratio compared to two-state. However, the first and second cache hit ratios are almost unchanged by increasing the states number greater than three.

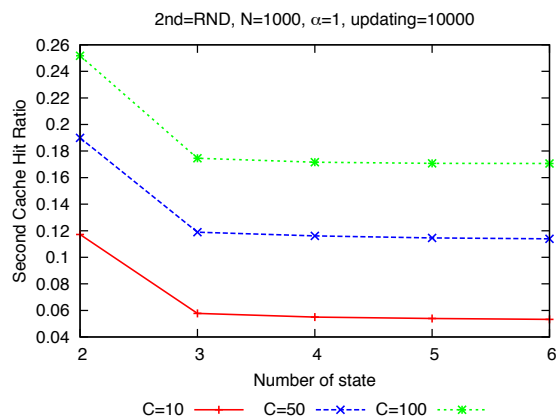


Figure 4.6: The second cache hit ratio with RND versus number of states with $N = 1000$, $\alpha = 1$

By comparing Figure 4.5 with Figure 4.6 and 4.7, we find that using the n -state at the first cache leads to the situation that the second cache obtains the highest hit ratio when it is managed by LRU. In addition, the second cache obtains almost equal hit ratios using FIFO and RND but less than LRU. This means that the hit ratio of LRU, FIFO and RND has the same order at the first and second caches. However, if the first cache is managed by replacement policies, the hit ratio order is not the same in both first and second caches. For example, if the first cache managed by LRU the order of hit ratio in the second cache is RND, FIFO, LRU (RND obtains the highest; FIFO obtains less than RND and more than LRU).

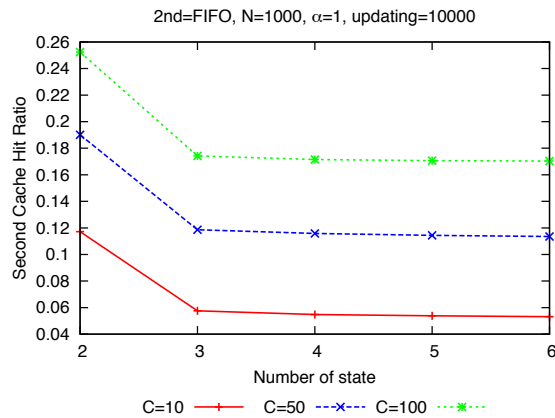


Figure 4.7: The second cache hit ratio with FIFO versus number of states with $N = 1000$, $\alpha = 1$

This property of same order implies that n -state only uses the locality of reference to obtain a high hit ratio as much as possible and provides the opportunity for the subsequent caches to obtain too. However, the replacement policies use the locality of reference to increase the hit ratio but destroy the locality of references for subsequent caches.

Trace-Based Workload

Findings and Discussion (Overall Hit Ratio vs n): Figure 4.8, 4.9 and 4.10 respectively show the second cache hit ratio of LRU, RND and FIFO while the first cache is managed by n -state. The main decrease happens when n is increased

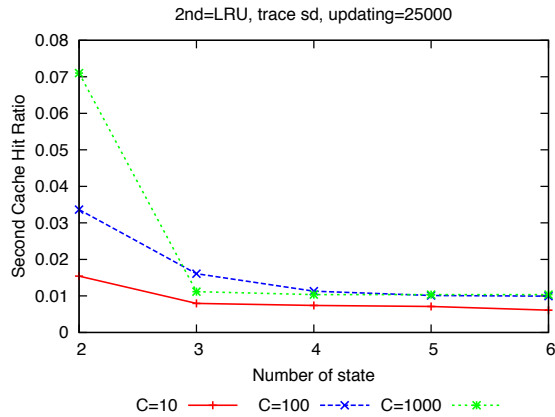


Figure 4.8: The second cache hit ratio with LRU versus number of states by sd trace

from two to three. In addition, the property of having the same trend of the hit ratio of LRU, RND and FIFO is also valid for cache size of 10 and 100. However, RND outperforms LRU for cache size of 1000 at the second cache because the

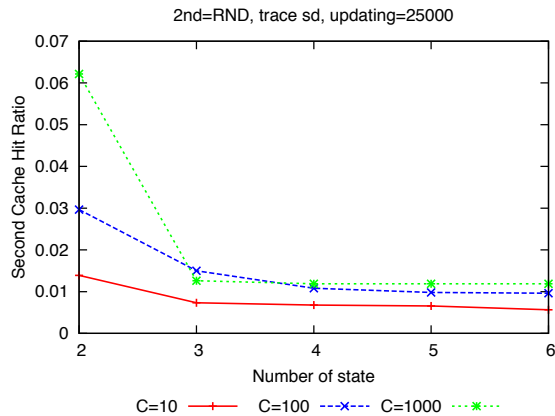


Figure 4.9: The second cache hit ratio with RND versus number of states by sd trace

updating period of 25000 requests. That is, large cache size leads to long time (large number of requests) for the cache to reach to the frozen state. Therefore, it is possible that the cache returns to the updating state before reaching to the frozen state and the first cache keeps replacing the missed contents. This causes the missed requests of the n-state to have the characteristics similar to replacement policies.

Based on the results in this section and in Appendix D, we conclude that

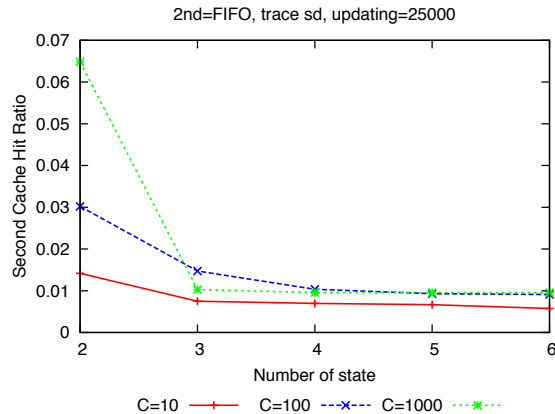


Figure 4.10: The second cache hit ratio with FIFO versus number of state by sd trace

$n = 3$ is the best practical option because i) it considerably improves the hit ratio of cache by dealing with the pollution ii) it provides the opportunity for subsequent caches to obtain high hit ratio iii) its implementation overhead is not considerable (two bits per slot). Therefore, we only present the results of three-state (3S) in our evaluation for the remaining sections of this chapter. In the next two subsection, we compare our three-state policy with LRU, LFU and two-state in terms of standalone and overall cache hit ratio under synthetic and real workloads.

4.1.4 3-State First Cache Hit Ratio

In this section, we compare the hit ratio of three-state policy with LRU, LFU and two-state. Although implementing of LFU is not practical for an ICN network of caches because of the current memory technology [5] and the large catalog size in the Internet, we select LFU because LFU is the optimal replacement policy under IRM [23].

Synthetic Workload

Setting: We measure the first cache hit ratio managed by four above mentioned policies versus cache size and α (popularity) where there are 1000 contents with the updating period of 50000.

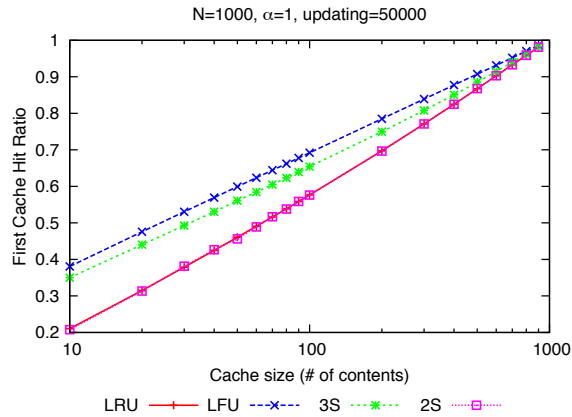


Figure 4.11: The first cache hit ratio with different policies versus cache size

Findings and Discussion (3-State Hit Ratio): Figure 4.11 and 4.12 show that three-state policy obtains higher hit ratio compared to LRU and two-state and pretty close to the hit ratio of the LFU (optimal for IRM). The same trend is also valid for all of the results presented in Appendix C. These results indicate that the three-state policy is able to capture the popular (high frequency) contents in the workload.

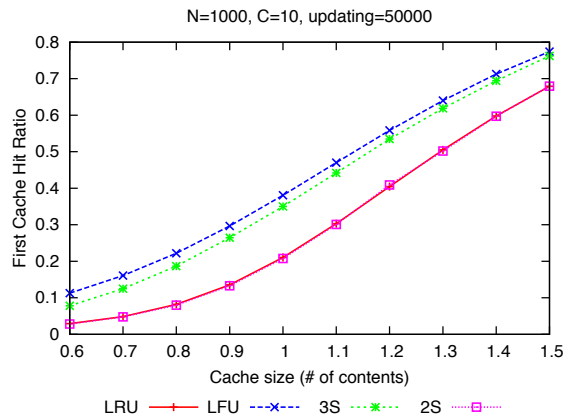


Figure 4.12: The first cache hit ratio with different policies versus α

Trace-Based Workload

Findings and Discussion (Hit Ratio under Different Traces): Figure 4.13 shows the first cache hit ratio for all traces (described in Section 3.1) for a specific cache size of 100 and updating period of 10000. As depicted in Fig-

ure 4.13, the three-state policy obtains the hit ratio pretty close to the highest hit ratio and it is always greater than the two-state hit ratio due to the property of removing the one-timer contents. As it can be seen, LRU outperforms LFU for *bo*, *pa*, *sj* and *sv* traces and LFU outperforms LRU for *ny*, *rt*, *sd* and *uc*. However, the three-state policy obtains hit ratio close to the highest hit ratio (either LRU or LFU) for all of the traces. This indicates that the three-state performs well by capturing the popular contents and the correlation between requests.

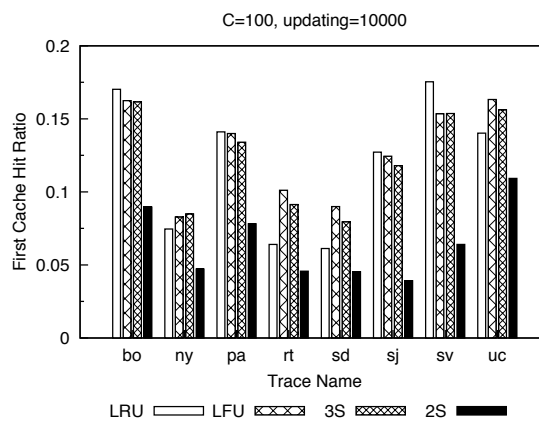


Figure 4.13: The first cache hit ratio with different policies and traces

4.1.5 3-State Benefits for Overall Hit Ratio

Synthetic Workload

Findings and Discussion (3-State Overall Hit Ratio): Figure 4.14 shows that the LRU at the second cache obtains the hit ratio of one with LFU at the first cache but less than one with two-state and three-state for $C \geq 500$. The 500 is the size that the summation of both cache sizes is equal to the catalogue size (1000). Therefore, if the first cache can always keep the same set of contents, the second cache obtains the hit ratio of 1 because the cache size is greater than or equal to the missed contents set from the first cache. This is what LFU does at the first cache by filtering the requests for the first $C - 1$ most popular contents and pass the rest. However, the two-state and three-state do not have this property because they do not keep the frequency of contents and their cached contents may

be different in two consecutive updating periods. This leads to some changes in the missed contents set of the two-state and three-state and consequently prevents the second cache from reaching the hit ratio of one.

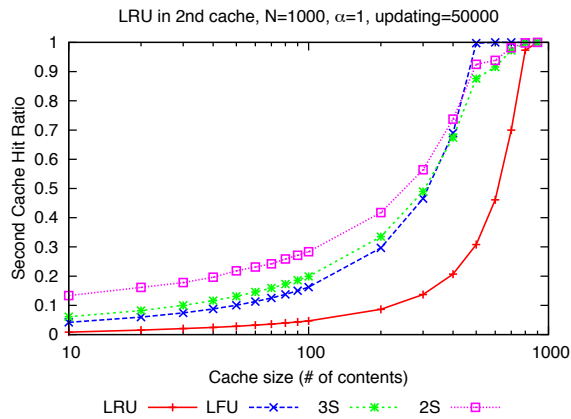


Figure 4.14: The second cache hit ratio with LRU, $N = 1000$, $\alpha = 1$

Figure 4.15 shows the LRU hit ratio in the second cache with 1000 contents and cache size of 10 versus different α . The figure shows that LRU at the second cache obtains the highest hit ratio while the first cache is managed by two-state. The subsequent ranks are achieved by three-state, LFU and LRU respectively.

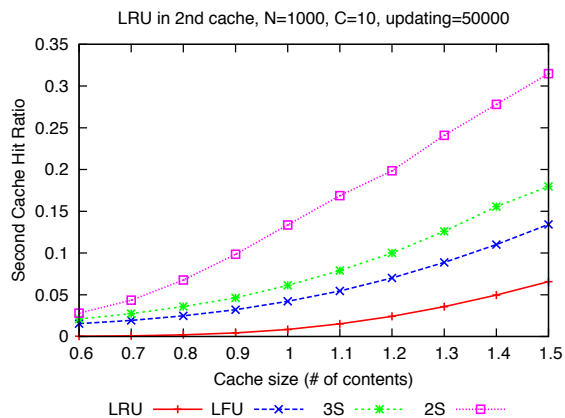


Figure 4.15: The second cache hit ratio with LRU, $N = 1000$, $C = 10$

The hit ratio of LRU, RND and FIFO in the second cache, presented in Appendix C, have the same trend for different combinations of cache size, catalog size and α .

Trace-Based Workload

Findings and Discussion (3-State Overall Hit Ratio): Figure 4.16 and 4.17 show that the second cache can obtain the highest hit ratio while the first cache is managed by the two-state. This is achieved for the real traces with the cost of low hit ratio at the first cache. However, the three-state policy is able to obtain a hit ratio close to the best policy at the first cache and it also provides opportunity for the second cache to obtain a high hit ratio. Figure 4.16 and 4.17 show that the second cache hit ratio is low for *ny*, *rt* and *sd* traces while the first cache is managed by three-state. This is due to the fact that three-state spends a long time before reaching frozen state and keeps replacing during the time. This prohibits the second cache from obtaining a high hit ratio. We solve this issue through the coordination where the high level idea is that caches helps each other to go to the frozen state faster than the case of a standalone cache.

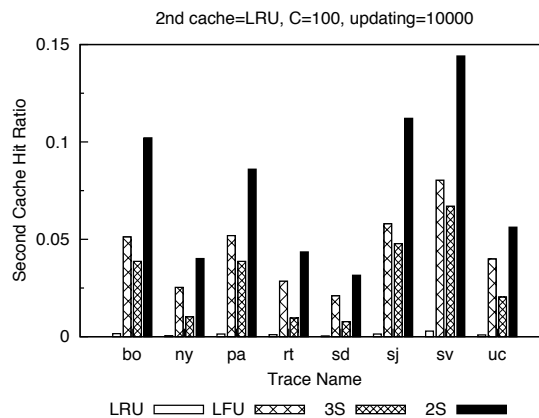


Figure 4.16: The second cache hit ratio with LRU and different traces

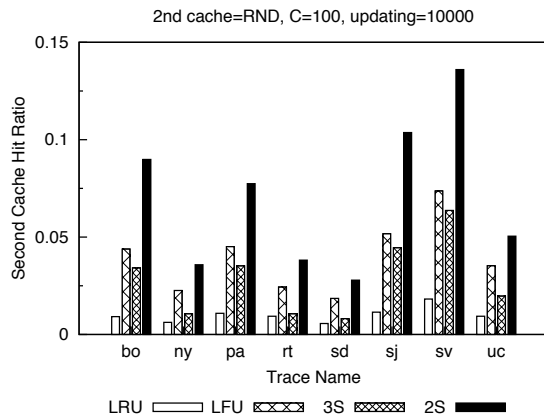


Figure 4.17: The second cache hit ratio with RND and different traces

So far, we have discussed our n -state policy that is able to deal with one-timer problem for $n \geq 3$ in addition to thrashing and contention. It obtains a high hit ratio for both synthetic and real workloads. In the next section, we complete our first approach for proposing the lightweight coordinated schemes for ICN network of caches by introducing our n -state coordinated scheme.

4.2 Coordinated N-State Scheme

As shown in the previous section, managing a cache with n -state policy provides the opportunity for subsequent caches to obtain high hit ratio. However, independently managing the caches in a network of caches leads to redundant copies in different caches. This reduces the overall hit ratio of network of caches. To solve this problem, we propose our lightweight coordinated scheme that is integrated with our n -state policy. First, we introduce three concepts used in this section to explain the idea. Then, we discuss our design principles, their intuitions and high level implementation ideas. Finally, we explain how our protocol implements the design principles.

4.2.1 Defining Path, Closeness Rank & Useless Duplicate

Three concepts used in our protocol are: i) path ii) closeness rank and iii) useless duplicate copy. A path consists of a set of routers and links that connect a group of consumers to a producer. For example, Figure 4.18 shows three different paths. With respect to a path, we define the concept of *closeness rank* of a router as the hop distance of that router from the consumers of the path. Therefore, the router with the smallest (largest) hop distance from the consumers of a path has the highest (lowest) closeness rank. A router may have different ranks if it is involved

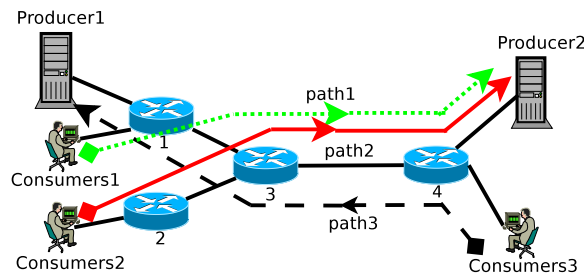


Figure 4.18: Definition of closeness rank based on path

in multiple paths with respect to each path. For example, router 4 in Figure 4.18 has rank 1 with respect to path 3 but has rank 3 with respect to paths 1 and 2.

The third concept, useless redundant copy, is related to the situation that multiple routers have the same copy of a content. For example, let us assume that multiple routers have the same copies of content X that are called duplicate copies. A copy of X in router Y is a useless duplicate if for each path towards X 's producer passing through Y there is a higher closeness-ranked router that has a copy of X . Therefore, router Y does not receive any request for content X and there is no point in keeping that useless copy. For example, let us assume that all three routers 1, 2 and 3 in Figure 4.18 cache content X of *producer2* that is requested by consumers 1 and 2 from paths 1 and 2. The duplicate copy of X at router 3 is *useless* because for each path towards *producer2* (path 1 and 2), there is one router that has X (router 1 and 2 respectively) with higher closeness rank compare to router 3. In contrast, if a duplicate copy is not useless, it is useful.

For example, the duplicate copies of X at router 1 and router 2 in our scenario are useful. After defining the three concepts used in our protocol, we describe our two design principles in the next subsection.

4.2.2 Design Principles

Our design principles are:

1. *Managing redundant copies:* In the whole network of caches, there should not be any useless duplicate copy in a frozen slot.
2. *Bringing popular contents close to consumers:* For each path, the router with the highest (lowest) closeness rank should have the highest (lowest) chance of caching the popular contents.

We achieve our design principles through three rules. First, when a consumer requests a content from a producer, only one of the path routers may write the content. Second, to write a content, a router in state i should receive the corresponding request and piggybacks the message of “*the corresponding content of this request is going to be cached if it is at least in state $i+1$* ” because the router is trying to transit to state $i+1$. Then, the router should announce this message to the subsequent routers along the path, which have lower closeness ranks. Third, these subsequent router(s) receiving a request with such a message cannot cache the corresponding content if the content has the minimum state of $i+1$. Moreover, if they have the content, they should fetch a new content instead of that content.

Through our three rules, we achieve the first design principle. Because of the first rule, every time that a content is fetched, it is only written to one router in the path. Therefore, the only way to have useless duplicate copy in a path is to write the same content in at least two routers of the path through multiple fetches and the order of writing has to follow a specific pattern due to the second rule. The pattern is that the router with lower rank must write the content before the higher ranked router otherwise it is not possible for the lower ranked to receive

the request (filtered by higher ranked). Without receiving a content request, it is not possible to write that content based on the second rule. Moreover, due to the third rule, the specific order lets the lower ranked router to distinguish the situation and fetch a new content instead of the duplicate copy.

The second design principle means that a router in the state i has the privilege to fetch the contents with minimum state $i + 1$ from all routers with lower ranks with respect to each path because it is highly probable that a content at least in state $i + 1$ is more popular than a content in state i . The second principle is guaranteed by our three rules because a router should receive the corresponding request for a content to be able to cache the content based on the second rule and a router receives a content request only if the path's routers with higher closeness rank have not cached the content. Therefore, a router cannot fetch content from the path's routers that have higher ranks but it can fetch from the lower ranked routers by announcing the specific message based on the second rule. By receiving such a message, the receiving router understands that it cannot cache the content based on the third rule and gives the priority to the higher ranked routers to cache the content.

4.2.3 Implementation

Our high level implementation idea is that each router, requiring to fetch a content, puts extra information (the minimum acceptable state of the content) in the request and forwards the request (*marking*). The provider (a router or producer) that serves the request decides about the location of the content based on the extra information (*deciding*). To do so, we use the extra fields in the request and content packet headers. We name each of these extra fields as *Distance from Candidate Router* (DCR) and there are $n - 1$ DCRs in an n -state coordinated scheme indexed from 1 (DCR_1) to $n - 1$ (DCR_{n-1}) representing the required content state of 1 to $n - 1$ (frozen). The value of a DCR in each router represents the distance from the candidate router in terms of number of hops. In addition, a

positive value of DCR_i in a router indicates that a higher ranked router is going to cache the corresponding content *if* it has the minimum state of i . On the path towards the producer, if a router needs to fetch a content in state i , the router changes the default value of DCR_i to 1 and forwards it. Other routers towards the producer increases the DCR_i by one until the request reaches the provider (a cache or producer). In the reverse path, each router decreases the DCR_i by one until the request reaches the candidate router with $DCR_i = 0$. On the other hand, a request with all $DCR_i = -1$ where $1 \leq i \leq n - 1$, set by consumers, indicates that the corresponding content is not going to be cached. We call the mechanism of changing the DCR value from -1 to 1 as *marking* a request.

When a cache in the state i receives a request, the request gets either missed or hit.

Missed Situation (marking rules)

1. Suppose that the request has not been marked by the higher ranked routers for fetching a content in state $j \leq i + 1$. That is, the state of all higher closeness-ranked routers is greater than i . The physical meaning of this situation is that the higher ranked routers are not interested in the contents that are in the slot state $i + 1$ or less. Therefore, the router is allowed to mark DCR_{i+1} to inform the lower ranked routers that it will cache the corresponding content if the content has the minimum slot state of $i + 1$.
2. Suppose that the request has been marked by higher ranked routers to fetch a content in the state $j \leq i + 1$. That is, the higher ranked routers are eager for the corresponding content if the content has minimum slot state of $j \leq i + 1$. Therefore, the receiving router is not allowed to mark this request and should increase the value of DCRs that are greater than zero and forwards the request.

These marking rules provide an important characteristic for a marked request. That is, if two routers have marked a request for the state i and j ($i < j$, $DCR_i > 0$

and $DCR_j > 0$), the higher (lower) closeness-ranked router must mark the request for state j (i). This is due to the fact that if the higher ranked router marks for state i , the lower ranked router could not mark the request based on marking rule two. This characteristic is used in deciding process when a provider determines the highest ranked router that has marked a request.

Hit Situation (deciding rules)

1. Suppose that several routers (with higher closeness rank) have marked a request. Therefore, there are multiple DCRs with value greater than zero. Then, there are two cases based on whether the content state meets the required minimum state of any router or not:
 - (a) *meets*: the provider decides that the content should be written to the router with the highest closeness rank among the routers of which the requirements are satisfied.
 - (b) *does not meet*: the provider decides that the content should be written to the router with highest closeness rank among the routers that have marked the request. In the special case of having only $DCR_{n-1} > 0$, the provider copies the value of DCR_{n-1} to DCR_{n-2} . This makes the content to get at least one hit in the candidate router before going to the frozen state.
2. Suppose that none of the DCRs of the request has been marked. In such case, the slot state of the hit content gets increased by one if it is not in the frozen and it is in a cache. Otherwise, the slot state remains constant.

Writing Only to One Router: The content provider (an intermediate router or a producer) guarantees that a requested content is only written to one router by only copying the decided DCR (determined by deciding rule 1) from the request header to the content header and setting other DCRs in the content header to -1 . However, when the DCR_1 is greater than zero, DCR_1 is also copied to inform the corresponding router to release the reserved slot. This is considered independent

from the deciding process because n-state uses reservation only in the updating state. In the path towards the consumer, when a router receives the request with DCR zero, it writes the content to its cache. In a special case if a router receives a content with $DCR_1 = 0$, the router checks whether there is another positive DCR or not. If there is, the router releases one reserved slot without writing the content and forwards the content because a higher ranked router is going to cache the content.

Managing the Duplicate Copies: The first case of the hit situation indicates that a hit content may be a useless duplicate copy in the future because a higher ranked router is going to cache the content. To solve this issue, the lower ranked router downgrades the slot state of the hit content to the current state of the cache ($n - 2$ when the cache is frozen). Therefore, the hit content will be replaced if it is a useless copy.

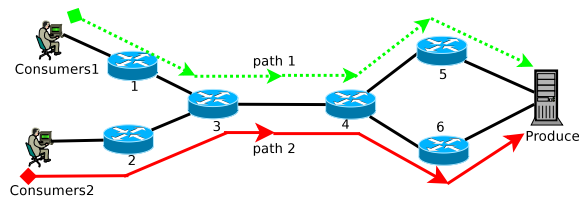


Figure 4.19: Managing the duplicate copies and multipath routing

The mechanism of downgrading guarantees the lack of useless redundant copies in frozen state under the assumption of lacking the multipath routing. To explain how we deal with the cases of having multipath routing, we use the example depicted in Figure 4.19 where there are two paths from router 4 to the producer. Assume that there are two different copies of content X at routers 5 and 6. In addition, assume that a request with at least one $DCR > 0$ gets hit at router 5 and the content X is going to be cached at router 3. Therefore, all of the future X 's requests get filtered by router 3 and the duplicate copy in router 6 becomes a useless copy (X in router 5 got downgraded). To solve this issue, we propose a mechanism that is only used by a router that uses multipath from itself to other nodes. For example, router 4 divides the requests targeting the producer through

routers 5 and 6. In our mechanism, router 4 forwards the requests having at least one $DCR > 0$ and targeting the producer to both router 5 and 6. Therefore, both routers downgrade their X copies and this prevents having useless duplicate copies in the frozen state. In the return path, the first X content packet that reaches router 4 will be forwarded and the later one will be discarded because there is no entry for the later one in the PIT.

Prioritizing Routers Based on Closeness Rank: Using our coordinated scheme, a cache in a path is able to fetch any content from caches with lower closeness ranks but not from the caches with higher closeness ranks. For example, router 1 in Figure 4.20 is able to mark any DCR of path 1 and caches the cor-

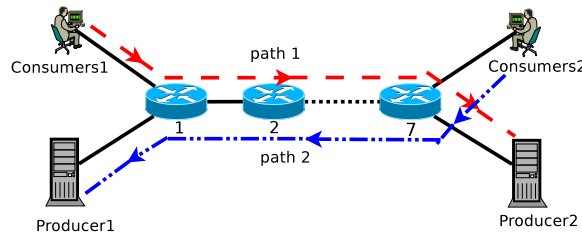


Figure 4.20: A linear topology with cross traffic

responding content if the content satisfies the state that router 1 is looking for. Router 1 can mark any DCR because it has the highest rank (1) for path 1 and receives the requests with DCRs of -1 directly from the consumers 1. On the other hand, router 1 has limitations for marking requests from path 2 because it has the lowest rank (7) for path 2 and the second path requests passed through six routers before reaching router 1. Therefore, router 1 is only able to mark a request from path 2 if the request has not been marked by router 7 to 2. Moreover, any request from path 2 that is marked by router 1 is served by producer 1 and will have state 1 because the contents in all producers have state one forever. Therefore, the content will have the lowest state and may be replaced with high probability.

Managing Packet Lost: Our coordinated scheme should deal with the situation that a request or content packet with a marked DCR_1 gets lost because a candidate router reserves a slot for that request. Lost of this kind of requests or contents

prevents releasing the reserved slot. To handle this situation, each router uses a timer called *reservation timer*. A router restarts a reservation timer at the time of i) marking a DCR_1 ii) receiving a content with $DCR_1 = 0$. In the case of reservation timer expiration, the router decreases the counter that represents the number of reserved slots and restarts the timer. The expiration period should be set relative to RTT of the network because a cache should wait at least RTT after marking a request to receive its corresponding content.

4.3 Evaluation

We evaluate our coordinated n-state scheme for $n = 2$ and $n = 3$ in Section 6.2 together with COCAP.

4.4 Summary

In this section, we extend our two-state policy to n-state policy that obtains higher hit ratio compared to two-state. Moreover, n-state inherits the advantages of two-state i) providing opportunity for other caches to obtain high hit ratios ii) adapting to the traffic pattern changes iii) simple implementation. We evaluate n-state based on synthetic and trace-based simulation and show that increasing n from two to three leads to a considerable improvement in hit ratio by capturing popular contents and removing one-timer contents. However, the improvement of increasing n for $n > 3$ is negligible.

In addition, we introduce a coordinated caching scheme integrated with n-state policy. Our coordinated scheme manages the duplicate copies such that there is no useless redundant copy in the frozen state. Moreover, it brings the popular contents close to the consumers based on each path. So far, we have completed our first approach for proposing the lightweight coordinated schemes for ICN network of caches. In the next chapter, we start our second approach by introducing a new cache management policy called CAP that is the base for our

second coordinated scheme, COCAP, which is presented in Chapter 6.

Chapter 5

CAP:

Contention-Thrashing-Pollution

Aware Replacement Policy

In this chapter, we start our second approach of proposing the lightweight coordinated schemes for ICN network of caches by introducing a class of replacement policies for managing a standalone cache. In the class of replacement policy, we manage a cache with two different policies for protected and unprotected segments. We explain how different combinations can use the advantages of three replacement policies, FIFO, RND and LRU, that are applicable in an ICN router [5]. Then, we discuss about our combination and explain how it can solve three caching problems at the same time and achieve a hit ratio comparable to other state-of-the-art. Then, we discuss the average time complexity and memory overhead of our implementation and prove that the average time complexity of our implementation is $O(1)$. Finally, we evaluate our CAP against the state-of-the-art policies designed for a standalone cache and show that CAP obtains close hit ratio to the state-of-the-art without keeping the meta-data of evicted pages.

5.1 A Class of Policies

Our main goal is to address the three predefined caching challenges: contention, thrashing and pollution without keeping the history of evicted contents. To achieve our goal, we **divide** a cache space into two variable-sized segments: protected and unprotected as depicted in Figure 5.1. The size of protected segment is S_p and the size of unprotected segment is $C - S_p$, where C is the cache size.

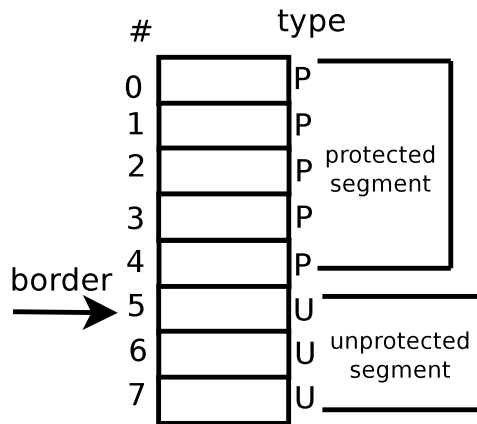


Figure 5.1: Protected and unprotected segments. $S_p = 5$ and $C = 8$

5.1.1 Cache Division

The idea of cache division between protected and unprotected gives us the opportunity to solve the pollution problem. Each segment is managed with an independent replacement policy and segments are separated by a pointer called *border*. We denote R_p and R_u as the replacement policies that manage the protected and unprotected segments, respectively. The cache starts its operation with $S_p = 0$ where all the cache slots are devoted to the unprotected segment. The missed contents are inserted into the unprotected segment based on R_u . When an unprotected content, i.e., a content in the unprotected segment, gets hit, our mechanism takes two actions: 1) the protected size is increased by one ($S_p = S_p + 1$) and 2) the hit unprotected content is moved to the protected segment and considered as a protected content. The location of the moved content in the protected segment

can be at the end or at the beginning of the segment depending on the implementation of R_p . On the other hand, if a protected content in the protected segment gets a hit, the R_p does the appropriate action. This process continues until the size S_p reaches a maximum threshold, S_p^{max} , and the mechanism restarts by setting all contents as unprotected contents and S_p to zero. The time between two consecutive restarts is defined as a *round*. A round is depicted in Figure 5.2 with two omitted cache states.

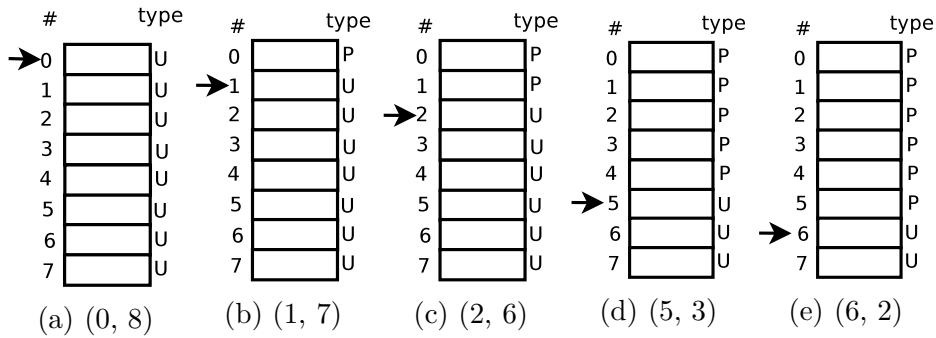


Figure 5.2: A cache *round* starts from 5.2a and ends with reaching 5.2e. $S_p^{max} = 5$ in this sample. The two intermediate cache states from 5.2c to 5.2d are omitted.

5.1.2 Adaptivity

Our cache policy uses a threshold, S_p^{max} , to limit the size of the protected segment. In fact, the protected size, S_p , grows from zero to its maximum of S_p^{max} and when S_p becomes greater than S_p^{max} , it is reset back to zero. The large value for S_p^{max} (close to C) leads to behaviors similar to LFU and large S_p^{max} performs well for the workload with no or low rate of popularity change. On the other hand, when the workload has a very high rate of popularity change, all of our policies described in Section 5.1.3 obtain their largest hit ratio pretty close to LRU that performs well for workload with high popularity change rate with $S_p^{max} = 1$. This is due to the fact that with $S_p^{max} = 1$, the policy's behavior is close to that of LRU, because the hit contents get moved to the MRU position and all contents could be considered as unprotected.

To adapt the policy between LFU and LRU based on workload, we use the number hits of the unprotected contents (N_u) and protected contents (N_p) during one round. Our logic is that the policy can push the S_p^{max} towards its minimum S_{min} as long as the unprotected segments can get more hits than the protected segment ($N_u > N_p$). Intuitively, $N_u > N_p$ happens when the protected contents do not stay popular for a long time because the workload popularity change rate is very high, i.e., after capturing a popular content, it becomes unpopular. On the other hand, the policy can push S_p^{max} towards its maximum $C - S_{min}$ as long as protected segments can get more hits than the unprotected segment, i.e., $N_p > N_u$. To dynamically adjust the maximum threshold, we update the value of S_p^{max} at the end of each round as follows:

$$S_p^{max} = \text{Max}(\text{Min}(S_p^{max} + N_p - N_u, C - S_{min}), S_{min}) \quad (5.1)$$

Consequently, S_p^{max} plays the role of adapting the size of the protected segment in the interval of $[S_{min}, C - S_{min}]$ when workload characteristics change.

Using S_{min} enables us to control the minimum size of both unprotected and protected segments. Without this bound if the cache decreases the S_p^{max} to zero, our policy is trapped and cannot have protected segment anymore. This is due to the fact that N_p is always zero by having $S_p^{max} = 0$. To avoid this situation, we set the S_{min} to $0.1 \times C$ in practice.

5.1.3 Choices of Replacement Policies (R_u, R_p)

We use three replacement policies, i.e., FIFO, RND and LRU, as our candidate replacement policies for protected and unprotected segments, because of their applicability for ICN routers [5], simplicity and wide range of characteristics. We investigate the advantages and disadvantages of different combinations of (R_u, R_p) under our adaptive policy framework. Since R_u is responsible for reacting to misses, in this situation, there is no difference between LRU and FIFO. Therefore,

Algorithm 1 CAP(R_u, R_p)

```

1: Input: The request stream of  $x_1, x_2, \dots, x_t, \dots$ 
2: Initialization:  $S_p^{max} = C - S_{min}, S_p = 0, N_u = 0, N_p = 0$ 
3: For every  $t > 1$  and  $x_t$ , only one case happens:

4: Case I:  $x_t$  gets hit in protected segment
5:  $R_p(x_t, hit)$  ▷  $R_p$  reacts to a hit for  $x_t$ 
6:  $N_p = N_p + 1$ 

7: Case II:  $x_t$  gets hit in unprotected segment
8:  $S_p = S_p + 1$ 
9: Move  $x_t$  to the protected segment
10:  $N_u = N_u + 1$ 
11: if  $S_p > S_p^{max}$  then
12:   ADAPT()
13: end if

14: Case III:  $x_t$  gets missed
15:  $R_u(x_t, miss)$  ▷  $R_u$  reacts to a miss for  $x_t$ 

16: function ADAPT( )
17:    $S_p^{max} = Max(Min(S_p^{max} + N_p - N_u, C - S_{min}), S_{min})$ 
18:    $S_p = 0$ 
19: end function

```

we consider to use either FIFO or RND for R_u . Since R_p is responsible for reacting to hits and FIFO and RND do not make any reaction to hit contents, we consider to use either LRU or NoOP for R_p , where NoOP denotes the mechanism that does not react to hits in the protected segment at all. Consequently, we consider four combinations of (R_u, R_p) : (FIFO, LRU), (FIFO, NoOP), (RND, LRU) and (RND, NoOP).

We discuss these four combinations from the perspective of the thrashing and pollution problems as follows. (FIFO, LRU) and (FIFO, NoOP) cannot handle thrashing because the unprotected segment is managed by FIFO that suffers from thrashing. However, the separation of unprotected and protected segments protects the popular contents from being polluted by one-timer contents. (RND, LRU) and (RND, NoOP) combinations can solve thrashing and pollution prob-

lems. Using RND in the unprotected segment gives the opportunity of getting hit for some contents in the presence of thrashing. These contents will be safe in the protected segment from the moment of moving to the protected segment till the end of a round. Moreover, the pollution can be solved by the separation between protected and unprotected segments.

So far, we have introduced our class of replacement policies and explained the advantages and disadvantages of different combinations. In the next subsection, we introduce the combination used for CAP and explain the reasons for selecting this combination.

5.2 CAP Combination: $(RND, NoOP)$

Although both combinations of (RND, LRU) and (RND, NoOP) can deal with thrashing and pollution, we choose (RND, NoOP) to further handle the cache contention problem. By using RND as the policy for the unprotected segment, multiple threads could write to the cache in parallel; by using NoOP for the protected segment, contention for the hit contents is in the slot level. Therefore, the choice of CAP decreases the granularity of contention from the whole cache level to slot level.

Although using RND in the unprotected segment may decrease the chance of getting hit for the contents in the unprotected segment, our investigation shows that most of the hits come from the protected segment. That is, the unprotected segment only has the role of detecting the popular contents. After detection, the contents are moved to the protected segment. Therefore, RND is the best option from our pool since it can deal with the thrashing problem and contention for missed contents at the same time.

So far, we have introduced and discussed our CAP policy in terms of its high level idea and advantages. In the next subsection, we explain the challenges from the implementation perspective and our solutions and we show that the average

time complexity of our CAP in the worst case scenario is $O(1)$.

5.3 CAP Implementation

In this section, we explain the issues of CAP implementation and our solutions. The first issue is the time complexity of our CAP implementation and we prove that the average time complexity of our policy can be as good as $O(1)$ depending on S_{min} . The second issue, *false protection*, is the possibility of considering an unprotected content as a protected content. Through approximation, we show that the probability of false protection is almost zero in practice.

5.3.1 CAP's Implementation Discussion

To implement (RND, NoOP), we need two variable-sized lists. The list for protected segment can be a traditional linked list that is easy to implement. However, the implementation of the unprotected list is more complicated because RND needs to have access to any location in the list in a constant time but accessing a random content in a linked list has the time complexity of $O(C)$. Another option for unprotected segment is to use the dynamic array. However, the wasted space of the dynamic array is also high [13], $\theta(C)$, where C is the total number of slots.

Instead of using linked list or dynamic array for the unprotected segment, we use a single array to keep both protected and unprotected contents. In addition, we maintain a variable v_i for each slot i . Related to v_i , we introduce a shared variable V_p among all slots in the cache. Both v_i and V_p have the length of b bits. If v_i equals V_p , the i^{th} slot is interpreted as a protected content (slot). Otherwise, the slot is considered as unprotected slot. At the time of writing a missed content that a cache needs to find an unprotected slot, a cache generates a random number k between 1 and C until finding $v_k \neq V_p$. At the time of getting hit for slot i , a cache copies V_p to the v_i . Each v_i is initialized to be 0 and V_p is initialized to be 1 when the CAP starts.

During a round, any missed content is brought to the unprotected segment, with its slot v_i set to be $V_p - 1$. When a content in the unprotected segment gets hit, its slot v_i is further increased to be V_p . At the end of each round, V_p is updated by $(V_p + 1) \bmod 2^b$, where b is the number of bits of V_p . Through this simple mechanism, the protected and unprotected contents can be distinguished.

However, there are two issues with this mechanism. The first issue is related to the time complexity of finding an unprotected content. The second issue is related to the possibility of considering a stale content, un-replaced and un-referenced, from 2^b previous rounds as a protected content in the ongoing round. For example, if a one-timer content comes to the cache in round i and is not replaced until round $i + 2^b$, the content can be considered as a protected content. This situation is defined as *false protection*. In the following subsections, we calculate the average time complexity in the worst case and the probability upper bound of having a false protection.

5.3.2 Time Complexity of CAP Replacement

The time complexity of finding an unprotected content is equal to the number of memory access before finding an unprotected content. We are going to calculate it in the worst case scenario that happens when the number of unprotected contents is minimum and the cache needs to find an unprotected content. That is, the number of unprotected contents in the cache is $S_{min} (C - (C - S_{min}))$ that happens when $S_p = S_p^{max}$ and S_p^{max} has its maximum value equal to $C - S_{min}$. In addition, the mechanism for finding an unprotected content is to generate a random number i between 1 and C until finding $v_i \neq V_p$. Therefore, the probability of finding an unprotected slot by generating one random number between 1 and C (in the worst case scenario) can be calculated by

$$p = \frac{S_{min}}{C} \tag{5.2}$$

Algorithm 2 CAP(*RND*, *NoOP*)

```

1: Input: The request stream of  $x_1, x_2, \dots, x_t, \dots$ 
2: Initialization:  $S_p^{max} = C - S_{min}$ ,  $S_p = 0, N_u = 0, N_p = 0$ 
3: For every  $t > 1$  and  $x_t$ , only one case happens:

4: Case I:  $x_t$  is found in the cache
5: if  $cache.v_i == V_p$  then                                     ▷ A protected content gets hit
6:    $N_p = N_p + 1$                                                ▷ Atomic increment
7: else                                                         ▷ An unprotected content gets hit.
8:    $S_p = S_p + 1, N_u = N_u + 1$                                ▷ Atomic increment
9:   if  $S_p > S_p^{max}$  then                                     ▷ A new round should be started
10:    ADAPT()
11:  else
12:     $cache.v_i = V_p$                                          ▷ Needs to lock the slot  $i$ 
13:  end if
14: end if

15: Case II:  $x_t$  is not found in the cache.
16:  $rindex = \text{Random}(1, C)$                                      ▷ Generate a random number between 1 and  $C$ 
17: while  $cache.v_{rindex} == V_p$  do
18:    $rindex = \text{Random}(1, C)$ 
19: end while
20: Copy content  $x_t$  into slot  $rindex$                          ▷ Needs to lock the slot  $rindex$ 
21:  $cache.v_{rindex} = V_p - 1$                                ▷ Needs to lock the slot  $rindex$ 

22: function ADAPT( )                                         ▷ ADAPT is inside a critical region.
23:   if  $S_p < S_p^{max}$  then                                   ▷ Executes only once at the end of a round.
24:     Return
25:   end if
26:    $S_p = 0$ 
27:    $V_p = (V_p + 1) \bmod 2^b$ 
28:    $S_p^{max} = \text{Max}(\text{Min}(S_p^{max} + N_p - N_u, C - S_{min}), S_{min})$ 
29: end function

```

Moreover, the number of times that is required to generate a random number (the number of memory accesses) has a geometric distribution with success probability of p . Therefore, the average number of memory access in the worst case, M_{avg}^w is equal to $\frac{1}{p} = \frac{C}{S_{min}}$ and can be calculated by

$$M_{avg}^w = 1 \times p + 2 \times p(1-p) \dots k \times p(1-p)^{k-1} \dots \quad (5.3)$$

which is

$$M_{avg}^w = \prod_{i=1}^{\infty} i \times p(1-p)^i = \frac{1}{p} = \frac{C}{S_{min}} \quad (5.4)$$

Therefore, the time complexity of the average memory access in the worst case is $O(\frac{C}{S_{min}})$. S_{min} plays an important role in balancing the rate of reaction to the workload changes and hit ratio of the protected segment. The larger (smaller) the S_{min} is, the lower (higher) hit ratio of the protected segment is and the faster (slower) reaction to the workload changes happens. We use $S_{min} = 0.1 \times C$ in our evaluation and obtain hit ratio close to ARC [57] for different workloads and cache sizes. By our configuration, the average number of memory accesses to find an unprotected content in the worst case is $O(10) = O(1)$. It should be mentioned that this is the average for the worst case and a cache is not in the worst case scenario all the time. Therefore, the average memory access in practice is less than 10 accesses.

5.3.3 Upper Bound of the False Protection Probability

The *false protection* happens when an unprotected content is considered protected because the content neither gets hit (may be one-timer) nor replaced for $2^b - 1$ rounds. This happens because the v_i starts from its minimum (zero) and reaches its maximum ($2^b - 1$ where V_p and v_i have b bits). Then, it restarts from zero and this process continues. Therefore, a content that is written or referenced in the round i may neither gets referenced nor replaced during next $2^b - 1$ rounds. Consequently, that content is considered as a protected content in round $i + 2^b$ and is called a *false protected content*. To calculate the upper bound probability of happening a false protection, we assume that this probability has its maximum when a one-timer content comes to a cache and does not get replaced (regular contents may get hit). Therefore, we find the upper bound of the probability for a one-timer content that does not get replaced in $2^b - 1$ rounds.

Let p_{kj} be the probability that the unprotected content in slot k gets replaced

through processing a single request by a cache that has j protected contents. This happens if the request gets missed, with probability m , and independently the CAP policy selects the cache slot k for replacement, with probability $\frac{1}{C-j}$ since one of the $C-j$ unprotected slots have to be selected uniformly. Thus,

$$p_{kj} = \frac{m}{C-j} \quad (5.5)$$

where m can be approximated by the cache miss ratio. Therefore, the probability that an unprotected content does not get replaced by processing a single request by a cache having j protected contents is obtained by

$$p'_{kj} = 1 - \frac{m}{C-j} \quad (5.6)$$

Let us assume that X_j denotes the number of requests that are processed when the cache has j protected contents in one round. Therefore, the probability that an unprotected content does not get replaced in one round, denoted by q , can be calculated by

$$q = \prod_{j=0}^{C-S_p^{max}} (p'_{kj})^{X_j} = \prod_{j=0}^{C-S_p^{max}} \left(1 - \frac{m}{C-j}\right)^{X_j} \quad (5.7)$$

and we have that

$$1 - \frac{m}{C-j} \leq 1 - \frac{m}{C} \quad (5.8)$$

Therefore,

$$q \leq \prod_{j=0}^{C-S_p^{max}} \left(1 - \frac{m}{C}\right)^{X_j} = \left(1 - \frac{m}{C}\right)^{\sum_{j=0}^{C-S_p^{max}} X_j} \quad (5.9)$$

The right hand side is decreasing in $\sum_{j=0}^{C-S_p^{max}} X_j$, which is in fact the total number of requests that are processed in one round. In one round, at least S_p^{max} requests should be processed and the minimum of S_p^{max} , as explained in Section 5.1.2, is S_{min} . Hence, the minimum of $\sum_{j=0}^{C-S_p^{max}} X_j$ is S_{min} , and

$$q \leq \left(1 - \frac{m}{C}\right)^{S_{min}} \quad (5.10)$$

Let $r(n)$ be the probability that a one-timer unprotected content does not get replaced during n consecutive rounds. $r(n)$ can be calculated by $r(n) = q^n$. Therefore, the upper bound of the probability that a false protection happens in a cache with b bits in v_i and V_p is

$$r(2^b - 1) \leq \left(1 - \frac{m}{C}\right)^{S_{min}} 2^{b-1} \quad (5.11)$$

This upper bound is less than 10^{-10} for cache size from 10 to 10^{10} , $m \geq 0.01$ and $S_{min} \geq 0.1 \times C$. The larger the m is, the smaller the upper bound is.

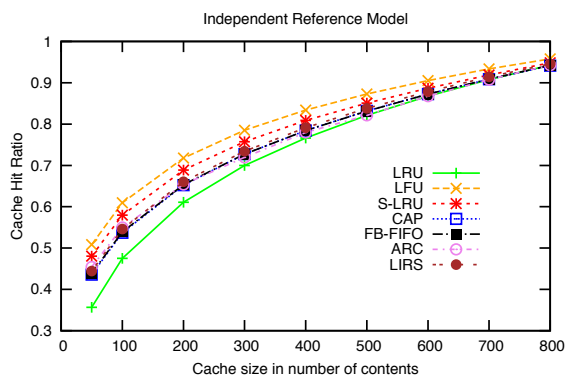
So far, we have explained our CAP with its advantages in terms of solving the three caching problems and its implementations. Furthermore, we have shown that our implementation is easy to be deployed in an ICN router. In the next subsection, we compare our policy with the state-of-the-art policies for standalone caches and show that CAP obtains comparable results without keeping any meta-data for evicted contents.

5.4 Evaluation

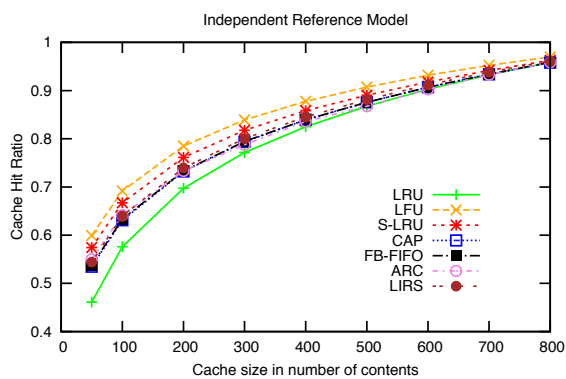
In this section, we evaluate our replacement policy under both synthetic and real workloads. Using the synthetic workload enables us to separately examine the effect of thrashing and pollution on CAP policy and other policies. We generate our synthetic workload based on Independent Reference Model (IRM) [23] assumption and add a different portion of polluting and thrashing traffic. In addition to synthetic scenarios, we use block-level traces collected by Microsoft Research Cambridge [61] as our real workloads. The detailed analysis of some of the traces can be found in [61, 93].

In both synthetic and real scenarios, we compare our CAP policy with LRU as the most common replacement policy, S-LRU as the pollution resistant version of LRU, LFU as the optimal replacement policy under IRM assumption, ARC [57] and LIRS [43] as the very good self adaptive policies and FB-FIFO [35]. It should

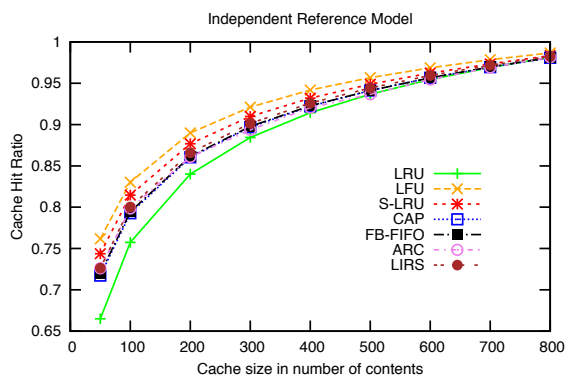
CHAPTER 5. CAP: CONTENTION-THRASHING-POLLUTION AWARE REPLACEMENT POLICY



(a) $\alpha = 0.9$

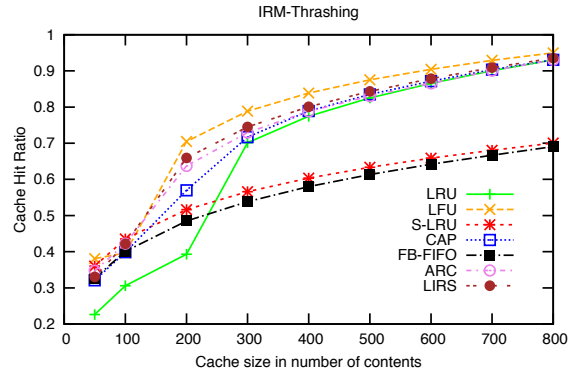


(b) $\alpha = 1.0$

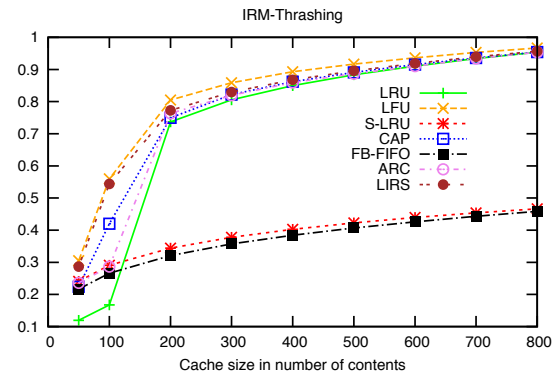


(c) $\alpha = 1.2$

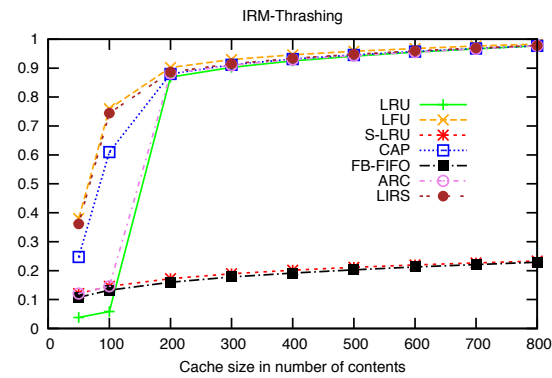
Figure 5.3: Cache hit ratio versus cache size under synthetic workload. Total number of contents in the system is 1000.



(a) $\gamma = 0.25$

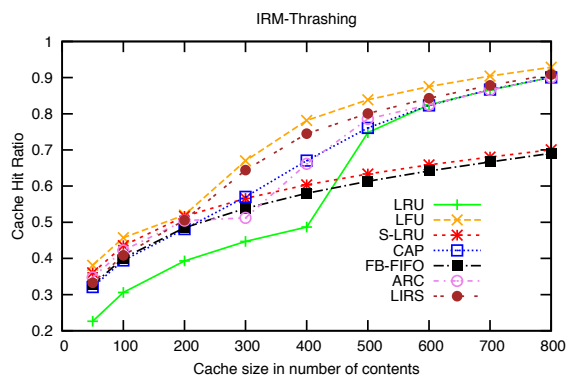


(b) $\gamma = 0.5$

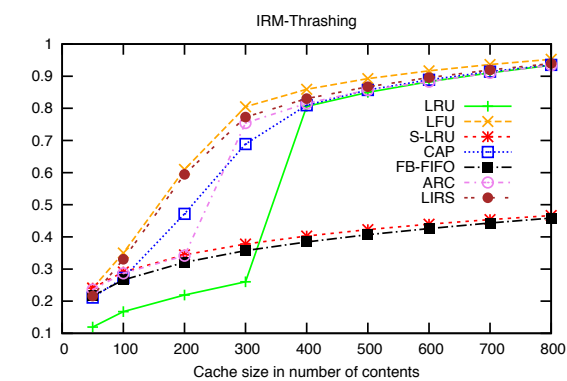


(c) $\gamma = 0.75$

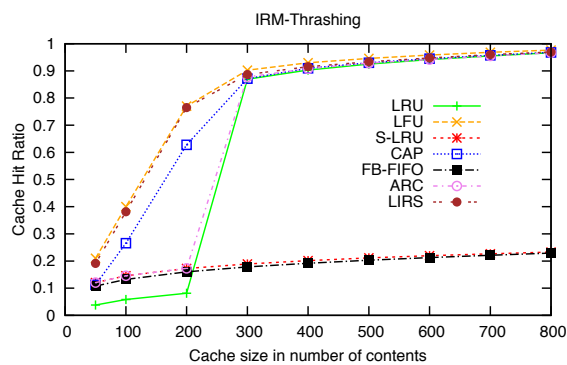
Figure 5.4: Cache hit ratio versus cache size under synthetic workload. Total number of contents in the system is 1000, thrashing length (β) is 100 with different portion of thrashing workload (γ).



(a) $\gamma = 0.25$



(b) $\gamma = 0.5$



(c) $\gamma = 0.75$

Figure 5.5: Cache hit ratio versus cache size under synthetic workload. Total number of contents in the system is 1000, thrashing length (β) is 200 with different portion of thrashing workload (γ).

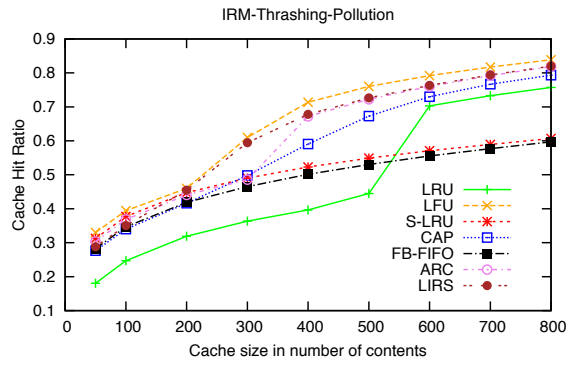
be mentioned that we exclude CAR [9] and Clock-Pro [41] from our comparison because their ancestors, ARC [57] and LIRS [43] respectively, that obtain higher hit ratios are present in our evaluation.

5.4.1 Synthetic Workloads

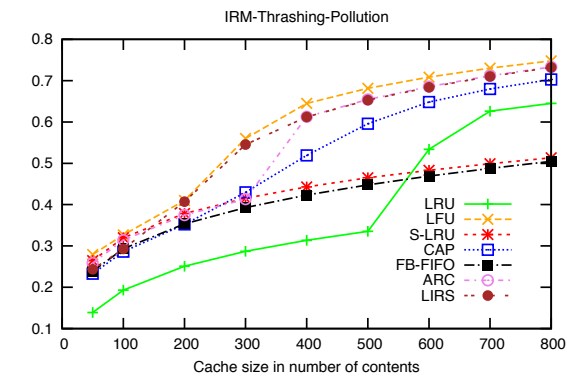
In this section, we evaluate different policies under IRM assumption in three sections. First, the requests are generated based on Zipf distribution with different Zipf slope of $\alpha = 0.9, 1, 1.2$. Second, we add a thrashing traffic to the Zipf generated workload. To do so, a portion of requests, γ , are generated in $(a_1, a_2, \dots, a_\beta)^*$ format where β is the length of the thrashing workload. Finally, we add a polluting workload to the Zipf and thrashing workload. This can be achieved by generating a portion of requests, Θ , in $(b_1, b_2, \dots, b_i, \dots)$ format where i is increased without any limit.

Independent Reference Model (IRM) We compare our policy with other policies under different configuration for Zipf distribution i.e., different total number of contents in the system and Zipf slopes (α). The results for three different Zipf slope are depicted in Figure 5.3. As it can be seen from the figure, LFU has the best (optimal) hit ratio under IRM and LRU has the worst hit ratio. S-LRU has the second rank among all replacement policies in terms of hit ratios. ARC, LIRS and CAP almost have the same performance. Therefore, CAP can capture the popular contents and obtain a comparable hit ratio with ARC and LIRS. Regarding the effect of the total number of contents, we have done the experiment with different total number of contents but we only present one set of results for 1000 contents due to the similar trends.

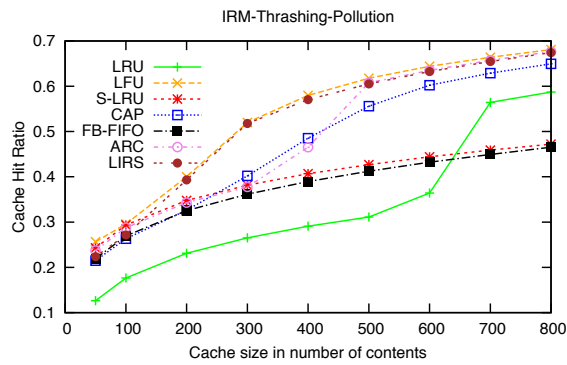
IRM with Thrashing In addition to the IRM traffic, there is a fraction of thrashing traffic, γ , with the thrashing length of β . Figure 5.4 shows the results for different γ where FB-FIFO and S-LRU have the worst performance because they use a smaller fixed fraction of cache to detect the popular contents and



(a) $\Theta = 0.1$



(b) $\Theta = 0.2$



(c) $\Theta = 0.3$

Figure 5.6: Cache hit ratio versus cache size under synthetic workload. Total number of contents in the system is 1000, thrashing length (β) is 200, portion of thrashing workload (γ) is 0.25 with different portion of pollution workload(Θ).

move them to their protected segments. On the other hand, LFU has the best performance since it has the access frequency of all contents and selects the most frequent contents to fill the cache. LIRS has the second rank in terms of cache hit ratio since LIRS has no bound for keeping meta-data of evicted contents (for example up to 4.5 times of the cache size[42]). However, CAP can obtain the hit ratio near to the best without keeping any meta-data.

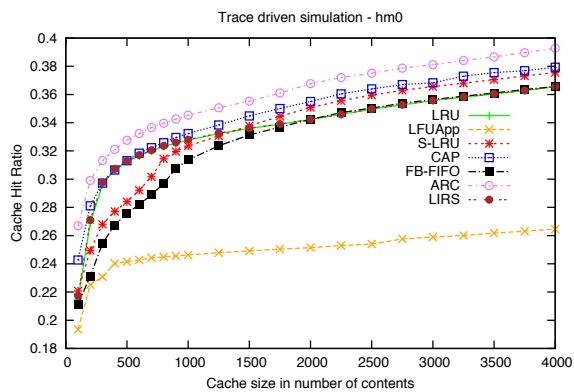
As it can be seen from Figure 5.4, there is a sharp jump for the hit ratio of ARC and LRU after point (100 contents) because ARC and LRU can obtain hits from thrashing workload for cache sizes larger than 100. In addition, the jump becomes sharper as the portion of thrashing requests increases because the effect of thrashing workload intensified. On the other hand, the sharp jumping point is shifted to 200 contents by increasing the length of thrashing workload, β , from 100 to 200 contents as depicted in Figure 5.5.

IRM with Thrashing and Pollution In the last set of experiment, we add the polluting workload to the combination of IRM and thrashing workloads. We do the experiment by different portions of polluting workload, Θ , of 0.1,0.2,0.3. As it can be seen from Figure 5.6, increasing the Θ intensifies the thrashing problem and shift the sharp jumping point from 500 to 600 for LRU and from 300 to 400 for ARC because the reuse distance of the thrashing workload gets increased when the one-timer contents interleave with thrashing workload.

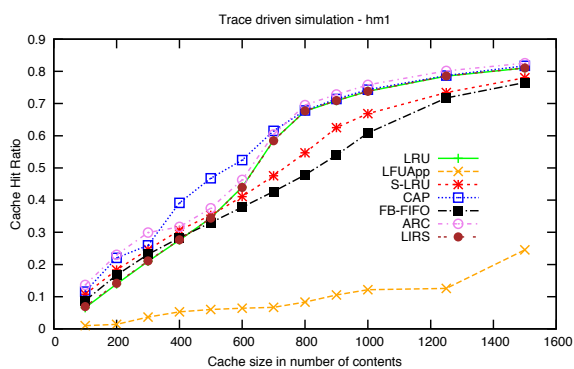
5.4.2 Real Workloads

We use Microsoft Research Cambridge (MSR-Cambridge) traces to evaluate our CAP policy. These traces are collected from the MSR-Cambridge data center servers and are in the block level [61]. We use 4 KByte block size for our experiments and simulate a content with one block. We compare CAP with the same group of policies described before except LFU because of its high overhead. Instead of LFU, we implement LFUApp in which a cache has a counter for each

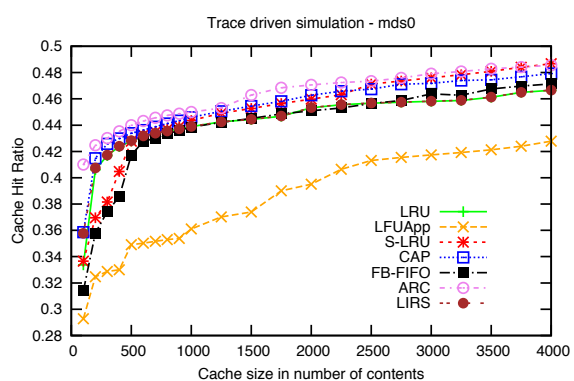
CHAPTER 5. CAP: CONTENTION-THRASHING-POLLUTION AWARE REPLACEMENT POLICY



(a) Hm0



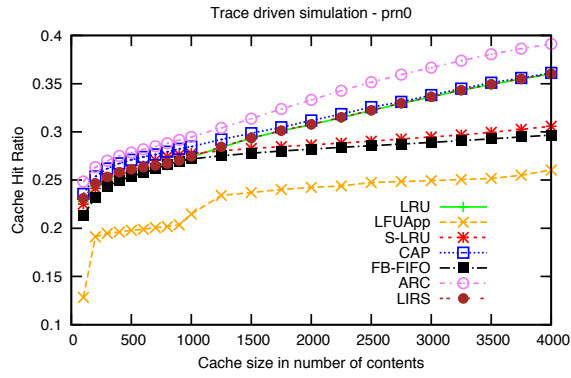
(b) Hm1



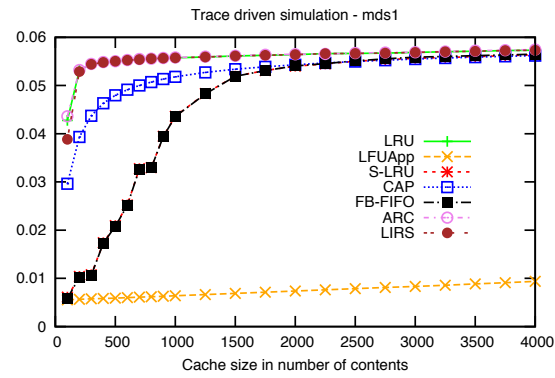
(c) Mds0

Figure 5.7: Cache hit ratio versus cache size under real workloads of hm0, hm1 and mds0.

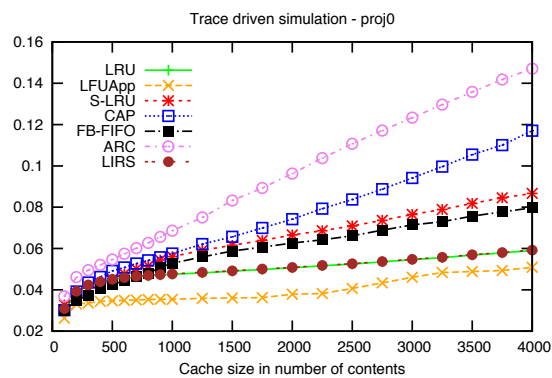
CHAPTER 5. CAP: CONTENTION-THRASHING-POLLUTION AWARE REPLACEMENT POLICY



(a) Prn0

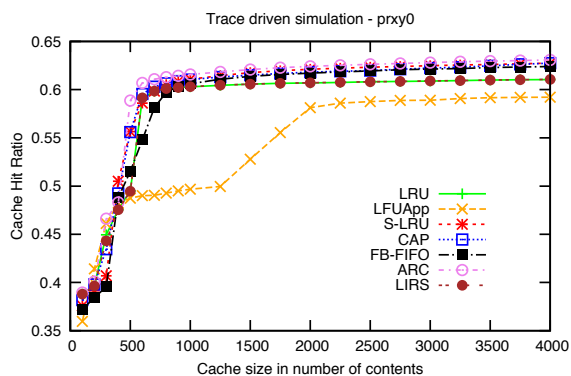


(b) Mds1

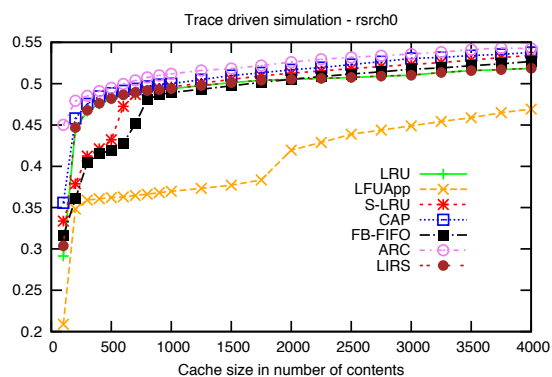


(c) Proj0

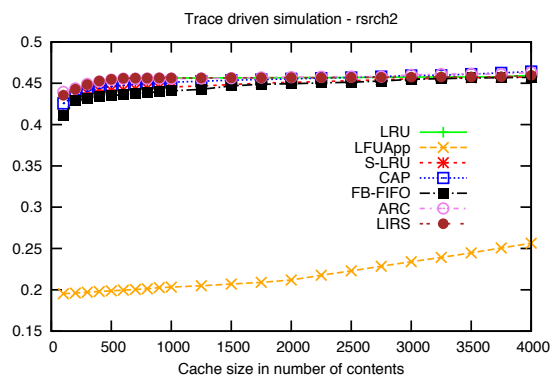
Figure 5.8: Cache hit ratio versus cache size under real workloads of prn0, mds1 and proj0.



(a) Prxy0



(b) Rsrch0



(c) Rsrch2

Figure 5.9: Cache hit ratio versus cache size under real workloads of prxy0, rsrch0 and rsrch2.

content that is in the cache instead of having a counter for each content that is in the system.

As it is depicted in Figure 5.7, 5.8 and 5.9, our CAP policy follows the hit ratio of the best policy (ARC) with different workloads. However, CAP does not need to keep the meta-data of evicted contents and its contention granularity is finer (slot level) than the ARC (cache level).

5.5 Summary

In this chapter, we introduce a new class of replacement policies by dividing a cache into two independently managed segments. The total size of the cache is adaptively divided between protected and unprotected segments. We explain different combinations of policies for protected and unprotected segments and their advantages and disadvantages. Among different combinations, we select a combination that could simultaneously solve the important caching problems of contention, thrashing and pollution. Our CAP policy decreases the granularity of contention to the cache slot and solves the thrashing and pollution problems by combining the advantages of different policies. Moreover, our policy adaptively reacts to the workload changes without any requirement for the history of evicted contents. CAP reaches the hit ratio close (sometimes better than) the hit ratio of the policies that use history of evicted contents. We show that our policy is simple to be deployed with average time complexity of $O(1)$.

So far, we have proposed our second replacement policy, base for our second co-ordinated scheme, that is simple to be deployed and it solves the caching problems. However, CAP cannot deal with the filter effect problem in network of caches. In the next chapter, we introduce the coordinated CAP, COCAP, that deals with the filtering effect by using the idea of freezing, borrowed from two-state policy, in a coordinated manner.

Chapter 6

Coordinated CAP Scheme

In this chapter, we complete our second approach for proposing the lightweight coordinated schemes for ICN network of caches by introducing the coordinated CAP scheme called COCAP that is based on the CAP introduced in Chapter 5. First, we explain the main idea of extending the CAP for a network of caches by making two unrealistic assumptions to make the transition smooth. Then, in two steps, we remove the assumptions and explain how our main idea can be implemented with a light overhead through four extra fields in the request and content packets and two variables in each router. Finally, we evaluate all of our coordinated schemes (COCAP, CO3S and CO2S) by comparing them with other schemes using synthetic and real topologies. We conclude this chapter with a summary. In this chapter, we use the concepts of *path*, *closeness rank*, *high (low) closeness-ranked routers* and *useless redundant copy* that are introduced in Section 4.2.1.

6.1 COCAP: Coordinated CAP Scheme

In this section, we explain the main idea of our COCAP by making two unrealistic assumptions i) having a centralized control with a holistic view ii) having a synchronized protocol among all caches. After describing the main idea through

example, we describe how to rectify the first assumption. Moreover, we describe the two important characteristics of our scheme and their implementations through examples. The first characteristic is the ability of caching the popular contents close to the consumers that decreases the content download time and transferred bytes. The second characteristic is to manage the redundancy to improve the overall network hit ratio. Finally, we explain how to remove the assumption of having synchronized protocol among all caches.

6.1.1 Centralized Control and Synchronized Caches

To explain the main idea of COCAP, we use an example of one cache managed as distributed caches depicted in Figures 6.1 and 6.2 respectively. Figure 6.1 shows a standalone cache with nine slots and Figure 6.2 shows a network of caches with nine slots of the standalone cache distributed among three routers. We manage

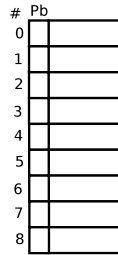


Figure 6.1: A single big cache

the network of caches as a single unified cache with the CAP policy. That is, we randomly write the contents that get missed in all caches to one of the nine slots and make a slot that gets hit protected until the number of protected slots reaches S_p^{max} . Then, all of the slots become unprotected and the process starts again. To do so, each cache slot should have one protection bit, Pb , to differentiate between protected and unprotected slots. In addition, we need a centralized control with a holistic view over all cache slots that synchronously become unprotected after the number of protected slot reaches S_p^{max} . These two assumptions, centralized control with a holistic view and synchronized caches, enable us to manage the three caches as a unified big cache that is managed by CAP policy. Consequently,

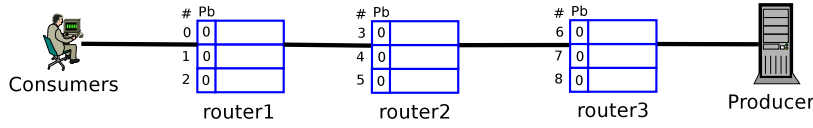


Figure 6.2: A big cache distributed in the network with a centralized control

there is no redundant copy in the network and the overall hit ratio is the same as a standalone CAP cache located between consumers and the producer. In addition, as it can be seen from Figure 6.2, we assign a *virtual slot number* with respect to the path to each cache slot. The virtual slot number starts from zero on the highest closeness-ranked router and is increased towards the lowest closeness-ranked router. We use this virtual slot number, defined per path, to explain how our COCAP can bring popular contents close to the consumers with respect to each path.

Popular Content Close to Consumers

As shown in Chapter 5 through trace-based simulation, CAP obtains hit ratio close to the state-of-the-art. Therefore, the unified big cache obtains high overall hit ratio and the probability of finding a popular content is similar in all caches because of uniform random number generation at the time of writing a missed content. However, in terms of content download time and transmitted byte, it is better to place the popular content close to consumers as CO3S obtains this property by prioritizing the routers based on their closeness ranks described in Section 4.2.2.

To cache the popular contents close to the consumers, we introduce the concept of *virtual cache* with respect to a path. A virtual cache is the first S_u^{vc} number of **unprotected** slots based on their virtual slot number with respect to a path plus the protected slots with the virtual slot number smaller than the virtual slot number of last unprotected slot. For example, consider Figure 6.2 that shows the network of caches after resetting all *Pbs*. In addition, let us assume that $S_u^{vc} = 3$. Therefore, the virtual cache is consists of the first three unprotected slots located

at the first cache. Moreover, let us assume that the virtual slot 1 gets hit and the network of caches status changes to Figure 6.3 where the virtual cache consists of the virtual slots 0, 1, 2, and 3 because the first three unprotected slots are 0, 2, 3 and the virtual slot 1 (protected and $1 < 3$).

With respect to the virtual cache, the missed contents from a path are only written to the unprotected slots. For example, the missed contents are only written

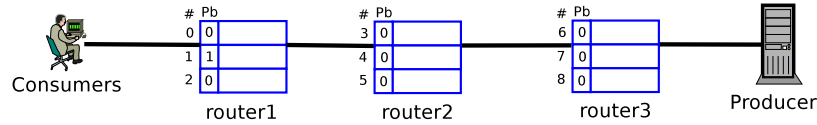


Figure 6.3: A virtual cache with size of 3

to slots, 0, 2 and 3 in Figure 6.3. As it can be seen from Figure 6.2 and 6.3, the virtual cache grows towards the producer but the maximum number of unprotected slots is the same. Through setting of S_u^{vc} , we can determine where the first hits happen. In our example, we guarantee that the first hit happens in the first cache and the content will be protected at the first cache. Moreover, the chance of getting the first hit for a popular content is higher than an unpopular content. Therefore, by using the virtual cache, we can protect the popular contents by converting their slots to protected from the highest closeness rank router towards the lowest closeness rank router with respect to a path. With respect to a virtual cache, we introduce the concept of *virtual hit* and *virtual miss*. A virtual hit happens whenever a request gets hit in a slot inside the virtual cache. Otherwise, it is considered as a virtual miss. For example, a virtual hit happens in Figure 6.3 if the contents in slots 0, 1, 2, 3 get hit but a virtual miss happens if the content in slot 6 gets hit. In addition, whenever a content gets a virtual hit, we set its Pb to one.

The centralized control with holistic view lets us to implement the virtual cache with respect to each path. For example, Figure 6.4 shows two paths crossing each other. For path 1, we write the missed contents only to slot 0, 1 and 2 until getting the first hit and grow the virtual cache towards the producer 2. However, for path

2, we write the missed contents only to slot 8, 7 and 6 (which have virtual slot number of 0, 1 and 2 with respect to path 2) until getting the first hit and move the virtual cache of path 2 towards the producer 1.

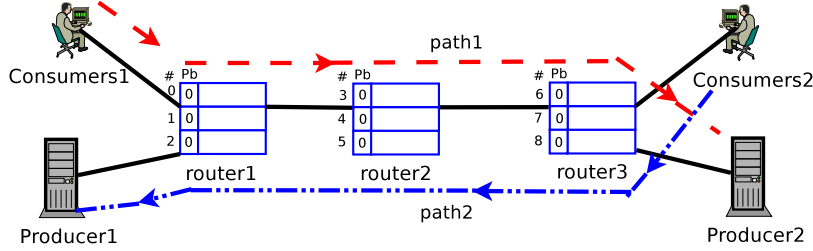


Figure 6.4: The virtual cache with respect to different paths

So far, we have introduced the main idea of our COCAP using examples under the assumption of having a centralized control with a holistic view and a synchronized protocol. In the next section, we explain that how we can implement our COCAP without the centralized control.

6.1.2 Synchronized Caches without Centralized Control

In this section, we release the assumption of having a centralized control with holistic view. Instead, we use a coordinated scheme, COCAP, to manage the virtual caches to obtain the same functionality. To do so, COCAP continues to replace the missed contents until all of the slots get protected and it synchronously resets the protection bits of all cache slots every updating period amount of time. We will remove synchronized assumption in the next section.

We implement our COCAP through piggybacking information on four extra fields in request and content packets and two variables in the routers. Before discussing the implementation, we need to explain the recently proposed variables and recently proposed. **Router variables:**

1. S_u represents the size of unprotected slots in a router. We define a cache as an *active* cache if its $S_u > 0$. Otherwise, the cache is inactive.

2. S_u^{vc} represents the number of unprotected slots in the virtual cache. S_u^{vc} of all routers are set to a constant and common value.

Extra fields in request and content packets:

1. Vb or virtual bit indicates whether the content gets virtual hit (1) or virtual miss (0).
2. DCR or Distance from Candidate Router is exactly the same as used in our previous coordinated scheme (described in Section 4.2.3). A packet with $DCR > 0$ indicates that the corresponding content of the packet is going to be written to another cache. Therefore, only the router that receives the content with $DCR = 0$ is supposed to write the content in the condition that the content gets virtual missed ($Vb = 0$). In addition, $Vb = 1$ indicates that the content gets hit inside the virtual cache and there is no need to be written to the virtual cache.
3. L represents the rank of the unprotected slot among S_u^{vc} unprotected slots to which the corresponding content is written. Therefore, it is an integer number between 1 and S_u^{vc} . For example, $L = 2$ for the cache in the state of Figure 6.3 where the $S_u^{vc} = 3$ refers to the virtual slot 3 because it is the second unprotected slot in the virtual cache.
4. L_{sum} of a request in a router represents the summation of the unprotected slots that are involved in the virtual cache in all higher ranked routers. For example, the L_{sum} of the request packet received by router 2 (3) in Figure 6.3 is 2 (3). The minimum of L_{sum} is zero and its maximum is S_u^{vc} . For example, let us assume that slots 0 and 2 get protected in Figure 6.3 (all slots of router 1 are protected). Then, the L_{sum} of the requests received by router 2 is zero because there is no slot involved in the virtual cache from router 1.

Implementation

When a router receives a request (receiving router), COCAP reacts based on whether the request either gets missed or hit:

Missed

1. $S_u = 0$. This means that all of the cache slots in the receiving router are protected (inactive cache). Therefore, the router only coordinates by increasing the DCR if $DCR > 0$ similar to coordinated n-state scheme.
2. $S_u > 0$ (an active cache):
 - (a) $L_{sum} = 0$. This means that the receiving cache is the first active cache involved in the virtual cache. We call this kind of cache as the *generator* because it generates an integer random number between 1 and S_u^{vc} as the value of L in the request. The value of L determines where the content should be written in the case of a virtual miss. Based on L , there are two cases:
 - i. $L_{sum} + S_u \geq L$. This means that the corresponding content of this request should be written in this cache in the case of a virtual miss (this router is the candidate router). Therefore, the router should mark the request by setting the DCR to one and it should change L_{sum} by $Min(L_{sum} + S_u, S_u^{vc})$.
 - ii. $L_{sum} + S_u < L$. This means that the receiving cache is not the candidate router. Therefore, the router only sets the L_{sum} by $L_{sum} + S_u$ and forwards the request.
 - (b) $0 < L_{sum} < S_u^{vc}$. This means that the receiving cache is involved in the virtual cache. Therefore, it should check whether it is the candidate router or not through Missed 2(a)i and Missed 2(a)ii.
 - (c) $L_{sum} = S_u^{vc}$. Although the receiving cache is active, the cache is not involved in the virtual cache for the request because the higher closeness-

ranked caches could provide the required number of unprotected slots. Therefore, the cache just coordinates through DCR.

Hit

1. $L_{sum} = 0$. This means that the router may be the first active router (generator) in the virtual cache depends on its S_u . Therefore, $L = 0$ for the request.

(a) $S_u = 0$. The cache is inactive and does not do anything.

(b) $S_u > 0$. The cache is active and needs to check whether the hit should be considered as a virtual hit or a virtual miss.

i. $L_{sum} + S_u \leq S_u^{vc}$ That is, the hit happens inside the virtual cache because the number of unprotected slots of the higher ranked caches plus the unprotected slots of the receiving cache is less than the virtual cache size. Therefore, the hit should be considered as a virtual hit. Moreover, if the protection bit of the slot is zero, the cache sets it and decreases its S_u by one.

ii. $L_{sum} + S_u > S_u^{vc}$ It is possible that the hit is not inside the virtual cache because the number of unprotected slots of the higher ranked caches plus the unprotected slots of the receiving cache is more than the virtual cache size. In this case, if the number of unprotected slots between the real slot zero and the hit slot (including) is less than the $S_u^{vc} - L_{sum}$, the hit is inside virtual cache and considered as a virtual hit. Otherwise, it is considered as a virtual miss. There are three cases:

A. $Pb = 0 \ \&\& \ \textit{inside} \Rightarrow Pb = 1$ and $S_u = S_u - 1$.

B. $Pb = 0 \ \&\& \ \textit{not inside} \Rightarrow$ relocate it with one item inside.

C. $Pb = 1 \Rightarrow$ do nothing.

2. $0 < L_{sum} \leq S_u^{vc}$ In this case, the receiving router is not the first active router (generator) but it may be involved in virtual cache depends on S_u .
 - (a) $L_{sum} + S_u \leq S_u^{vc}$. The hit is a virtual hit. Similar to Hit 1(b)i.
 - (b) $L_{sum} + S_u > S_u^{vc}$. This is the case that needs to investigate if the hit is a virtual hit or a virtual miss. Similar to Hit 1(b)ii.
3. $L_{sum} = S_u^{vc}$ That is, the receiving cache is not involved in the virtual cache and this hit is considered as a virtual miss.

So far, we have explained the COCAP implementation without centralized control with holistic view. However, we still count on the assumption of having synchronized network of caches. In the next section, we release this assumption.

6.1.3 Removing Synchronized Assumption

We assume that all caches synchronously clear their Pbs after updating period amount of time. However, the requirement of our coordinated scheme is that when an edge router resets its Pbs , the core routers with lower closeness rank involved in all of the paths passing through that edge router also reset their Pbs . To do so, we can use a limited flooding algorithm that is driven by the edge routers to reset the Pbs of all caches periodically. To achieve this, we set the edge routers as the driving routers with a predefined updating period. The driving routers reset their Pbs and forward a *reset* request to the routers involved in all of their paths. The routers, receiving the reset request, also reset their Pbs and broadcast the request in the condition that they have not received a reset request within a specific threshold amount of time. The threshold should be set in the order of the average network RTT. Moreover, we can use the Network Timing Protocol (NTP) [69] (in operation since 1985) to synchronize the network of caches. The NTPV3 has the resolution of one nanosecond and it is improved in the latest version, NTPV4 [63].

6.2 Evaluation

In this section, we evaluate our coordinated schemes, coordinated two-state (CO2S), coordinated three-state (CO3S) and coordinated CAP (COCAP), from the perspective of consumers, producers and ISPs for both synthetic and real network topologies.

6.2.1 General Setting

Comparing Schemes

Although, there are many coordinated schemes in the literature, some of them are not applicable to the ICN network of caches. For example, the schemes that are proposed in [68, 38, 51] measure the popularity of the contents. This makes the schemes impractical for an ICN router that should work at line speed [5]. Moreover, [27, 28, 85, 37] announce the contents of each cache to other caches but the small cache size compare to the catalogue size in ICN leads to a high rate of replacement and high communication overhead, which makes them impractical. Finally, schemes such as [79] requires a holistic view of all caches and this requirement is not practical with the ICN scale.

We compare our schemes with four schemes that are applicable in the ICN and representatives of other schemes. The first scheme is LRU universal caching (Leave Copy Everywhere, LCE) that is the base for evaluation of many schemes [67, 19] because it produces the results without any coordination. The second and third schemes are Leave Copy Down (LCD) and Move Copy Down (MCD) described in Section 2.2.3. LCD only writes the content into cache if it gets hit in the previous cache when the content is on the way back to the consumers. LCD is a representative for [19] because both approaches write the content missed in the network to the farthest router from consumers and move the contents towards the consumers. We also consider MCD that is similar to LCD except that the hit contents get erased from the cache. Consequently, MCD keeps the number of

redundant copies smaller than LCD. Finally, we compare our schemes with UNI, used in [19], that caches each content with the probability of $\frac{1}{hop-1}$ where the *hop* is the number of hops between the consumer and the current location of the content (excluding). For example, let us assume that a request gets hit after three hops from consumers in a cache (consumers \rightarrow router1 \rightarrow router2 \rightarrow router3). Therefore, the content is written to router2 and router 1 with probability of 0.5. The UNI is the representative of the schemes such as [67, 15] with the goal of writing a missed content in one of the caches on the way back to the consumers.

In addition to the four above-mentioned schemes, we compare our results with the optimal policy for linear topology. The optimal policy is defined as the policy that leads to the maximum overall hit ratio and also places the contents from most popular to the least popular from the closest router to the farthest router to the consumers. For a linear topology, the optimal policy is to select C_{all} most popular contents where C_{all} is the summation of all cache sizes. Then, we should place the selected contents from the most popular to the least popular into the routers with the highest closeness rank towards the lowest close rank. Although finding the optimal policy for linear topology is simple, it is an np-hard problem [80] for complicated topologies.

Content and Packet Generation

For request generation in the content level, we use the Poisson distribution because of the observation that shows the session level of Internet traffic is well modeled by a Poisson process [18]. Moreover, we use window-based request generation at the packet level on the consumer side. This window-based request generation starts with $w = 1$ and uses TCP rules such as increasing the window size by getting the packet and dividing window size by two for each lost packet. Finally, we should mention that we change the popularity in all of our experiments based on the approach described in Section 3.3. We use $p_s = 0.05$ and $\lambda_c = 0.00066$ for all experiments. Otherwise, we will explicitly mention.

Metrics

We define six metrics from three different perspectives of producers, consumers and ISPs. **Producers' perspective:** i) overall hit ratio, Hit_{Net} , is defined as $Hit_{Net} = \frac{R_{hit}}{R_{entered}}$, where $R_{entered}$ is the total number of requests entered to the network of caches and R_{hit} is the total number of requests get hit in the routers. **Consumers' perspective:** ii) average content download time that represents the consumers' average experienced delay for downloading contents. **ISPs' perspective:** iii) traffic reduction ratio, T_{red} , is defined as $T_{red} = \frac{T_{cache}}{T_{no-cache}}$ where T_{cache} ($T_{no-cache}$) is the total transmitted traffic with (without) caching. It should be mentioned that the transmitted traffic between consumers and edge routers are excluded from T_{cache} and $T_{no-cache}$ since caching does not affect this part of traffic. iv) average eviction rate per cache slot, E_{avg} , is defined as $E_{avg} = \frac{E_{total}}{S_{total} \times T_{sim}}$ where E_{total} is the total number of evictions in the network, S_{total} is the total number of cache slots in the network of caches and T_{sim} is the simulation time. E_{avg} implicitly represents the network energy consumption caused by the evictions in all caches. Finally, we use two more metrics to investigate the hit ratio and filter effect more precisely: v) average edge router hit ratio and vi) average core router hit ratio.

6.2.2 Linear Topology

Setting: To show the importance of removing the useless duplicates and prioritizing the caches based on their closeness rank, we start with a linear topology depicted in Figure 6.5.

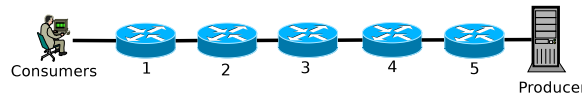


Figure 6.5: A linear topology with one-way traffic

There are 1000 contents on the producer and each content consists of one packet. In addition, the consumers generate the request with the rate of 20 re-

quests per second. The popularity distribution has the Zipf slope of one ($\alpha = 1$). We run the simulation for 20000 seconds. The updating period for CO2S, CO3S and COCAP is 750 seconds. For COCAP, we use the virtual size of 5 for $C = 5$ and virtual cache size of 10 for $C \geq 10$.

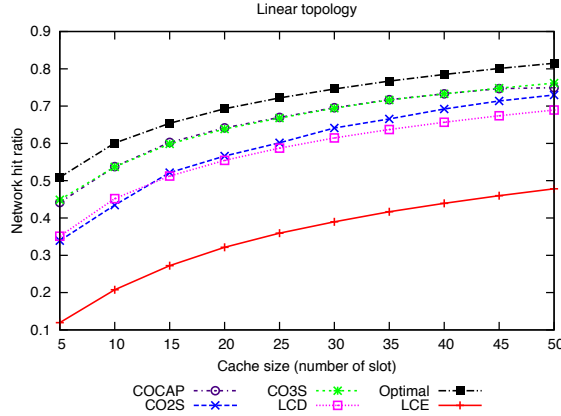


Figure 6.6: Overall network hit ratio

Performance versus Cache Size

Findings and Discussion: Figure 6.6 shows that the CO3S and COCAP obtain the overall hit ratios that are close to the optimal overall hit ratio. Although, the CO3S obtains lower edge router hit ratio compared to COCAP and LCD as depicted in Figures 6.7, it compensates on the core router hit ratio as depicted in Figure 6.8. The optimal obtains the highest hit ratio at the edge router but less than COCAP, CO3S and CO2S at the core router. This is expected because the optimal policy performs better than these schemes at the edge router. Therefore, the opportunity for obtaining the high hit ratio for core routers with optimal at the edge router is less than the situation that COCAP, CO3S and CO2S operate at the edge router. Moreover, optimal always obtains the highest overall hit ratio.

As it can be seen in Figure 6.7, the COCAP obtains higher hit ratio at the edge router than the CO3S. This is due to the way that COCAP and CO3S fill the edge router. In COCAP, all caches reset their P_b s together and the edge router is part of the virtual cache until all of its slots become protected. During this time, all of

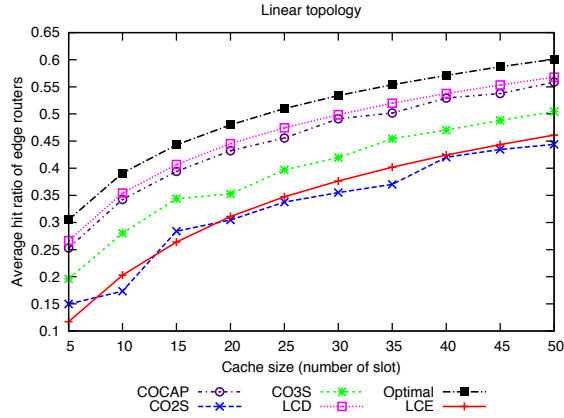


Figure 6.7: Average hit ratio of edge routers

the missed contents are written to the virtual cache and the contents that get hit out of virtual cache are considered as missed contents (virtual miss) and only the contents that get hit in the virtual cache become protected. On the other hand, the edge router with CO3S goes to updating state independent from other caches and gets help from them to go to frozen state. Other caches can be in frozen state and help the edge router by delivering their frozen contents to the edge router. That is, some of the contents are frozen in other caches and CO3S brings them to the edge router. Therefore, the condition for a content to be frozen at the edge router by CO3S is easier than a content to be protected by COCAP and it is more likely to have a content, not very popular, in an edge router managed by the CO3S than the COCAP. In contrast to the hit ratio of the edge router, CO3S obtains higher hit ratio than COCAP at the core routers, shown in Figure 6.8, because the popular contents that could not be placed at the edge router are placed at the core router by CO3S. Consequently, COCAP and CO3S have almost the same overall hit ratio.

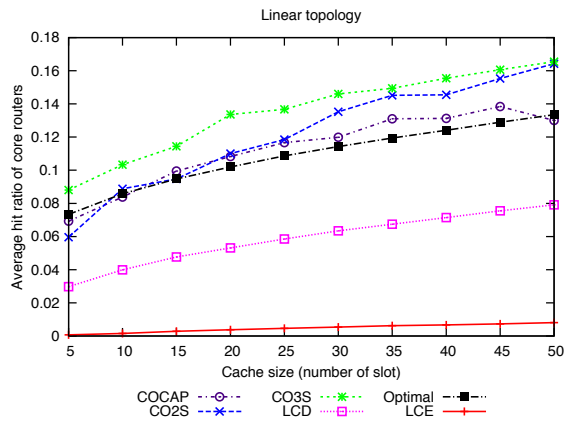


Figure 6.8: Average hit ratio of core routers

Comparing the LCD and CO2S in Figure 6.6, 6.7 and 6.8 shows that LCD outperforms the CO2S for edge router hit ratios but CO2S compensates by obtaining the higher hit ratio at the core routers. Consequently, both obtain the similar overall hit ratios.

Performance versus Popularity

Setting: In this section, we repeat the same experiment of the previous section for the cache size of 10 with different Zipf slope, α , to investigate the effect of varying the popularity. We choose cache size of 10 because in ICN network of caches the ratio of cache size to the catalogue size is very small.

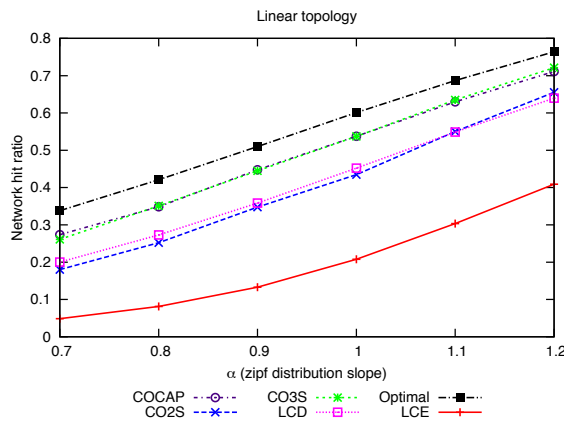


Figure 6.9: Overall network hit ratio

Findings and Discussion: As it can be seen from Figure 6.9, the COCAP and CO3S have the second rank for the overall hit ratio after the optimal. Similar to the previous section, CO2S and LCD have close overall hit ratios. Therefore, the trends discussed in this experiment are similar to the previous experiment.

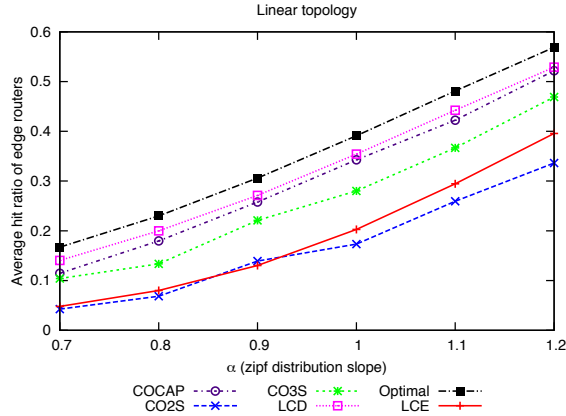


Figure 6.10: Average hit ratio of edge routers

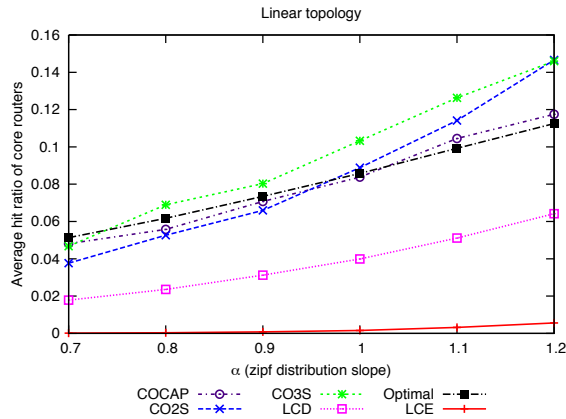


Figure 6.11: Average hit ratio of core routers

6.2.3 Cross Traffic

Setting: To show the importance of prioritizing routers based on their closeness rank with respect to different paths, we set an experiment by using a simple linear topology with presence of cross traffic depicted in Figure 6.12. The consumers on the left only generate requests to obtain the contents on the right producer (P_2)

while the consumers on the right only generate requests to obtain the contents on the left producer (P_1). We measure the overall hit ratio and the hit ratio of contents of P_1 and P_2 on each router. The optimal policy with cross traffic divides

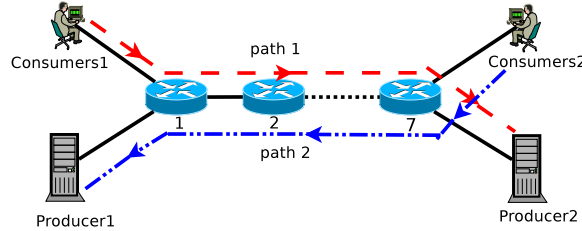


Figure 6.12: A linear topology with cross traffic

the overall cache size equally between the contents of the two producers because the consumers generate the requests with the same rate of 20 requests per second and the popularity slope of both producers are one ($\alpha = 1$). In addition, we place the content of P_1 (P_2) based on their popularity from router 7 (1) towards router 4 (4). There are 1000 contents (each has one packet length) on each producer. We run the simulation for 20000 seconds and the virtual cache size for COCAP is 10 slots.

Findings and Discussion: Figure 6.13 (Figure 6.14) shows the cache hit ratio

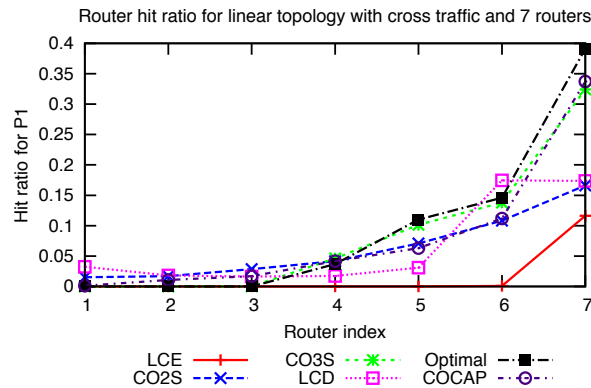


Figure 6.13: Individual hit ratio of P_1 with cache size of 10 and catalog size of 1000

of the contents on the left (right) producer, P_1 (P_2), requested by the consumers on the right (left) while the cache size is 10 (the ratio of cache size to the catalog size is 10^{-2}). Our CO3S and COCAP obtain the hit ratio close to optimal for

each router. As it can be seen in Figure 6.13, router 1 to 3 are not involved in caching the contents of P_1 for optimal because these routers have higher closeness

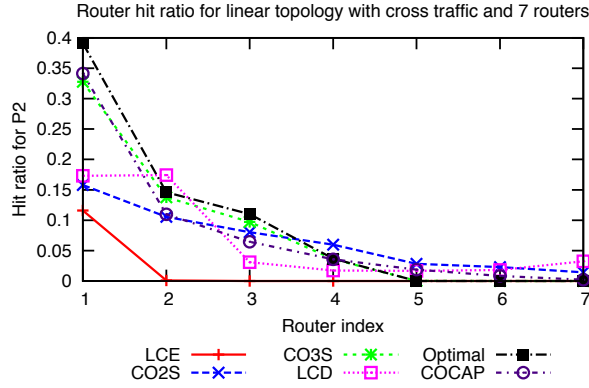


Figure 6.14: Individual hit ratio of P_2 with cache size of 10 and catalog size of 1000

rank for path 2 than path 1. Similarly, these routers are negligibly involved for COCAP and CO3S. This shows that COCAP and CO3S follow the optimal in terms of assigning the cache to different paths for linear traffic with cross traffic.

Figure 6.13 and 6.14 also show that COCAP and CO3S outperform LCD even for the edge cache hit ratio (in router 7 and 1 for path 1 and 2 respectively). This contradicts with the finding from linear topology without cross traffic in the previous section. The reason is that LCD writes the contents that get missed in all routers only to the last router. For example, LCD writes entire network missed contents of path 1 (path 2) to router 7 (router 1) and pushes them towards the consumers if they get hit. This property has a destructive effect on the cache hit ratio of the LCD on the first router on each path.

Figure 6.15 shows that CO3S and COCAP obtain the overall cache hit ratio pretty close to optimal. For example, CO3S obtains 5.4% less overall hit ratio on average than the optimal with the standard deviation of 10^{-3} .

6.2.4 Binary Tree Topology

Setting: In this section, we use the topology depicted in Figure 6.16 to investigate the effect of combining similar traffic patterns. That is, every higher level of router

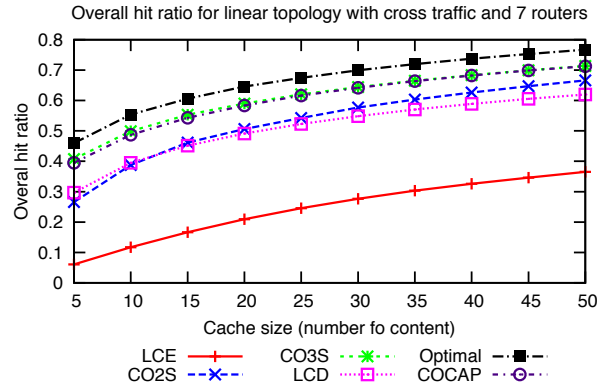


Figure 6.15: The overall hit ratio versus cache size for catalog size of 1000

receives the requests for the same set of contents from two underlying routers. There are 1000 contents on the producer. Each group of consumers generates the requests with the average rate of 5 requests per second and the Zipf slope is

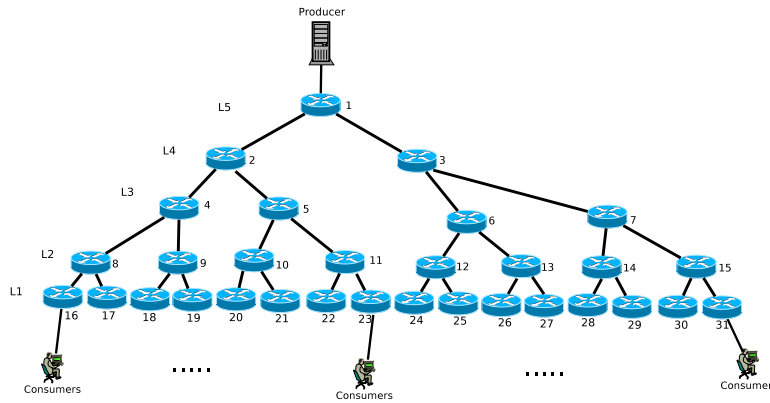


Figure 6.16: The binary tree topology

one ($\alpha = 1$). The popularity changes by the same way and setting described in Section 3.3 with $p_s = 0.05$ and $\lambda_c = 0.0008$ (period of 1250 seconds). We run the simulation for 6000 seconds. The updating period for CO2S, CO3S and COCAP is 500 seconds. The content size is based on the geometric distribution [33] with average content size of 500 packets. Consequently, there are 5×10^5 chunks in the networks. Finally, the COCAP virtual cache size is 500.

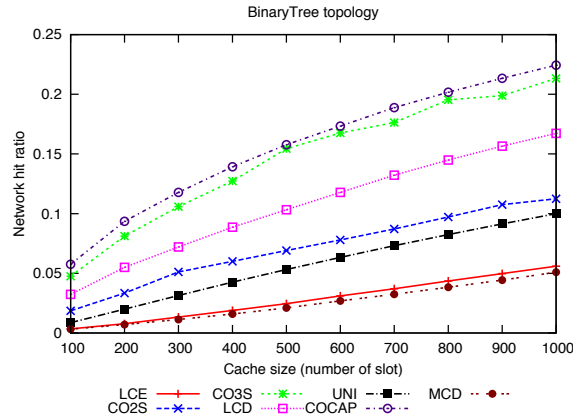


Figure 6.17: Overall network hit ratio

Performance versus Cache Size

Findings and Discussion: 1) **Overall hit ratio:** Figure 6.17 shows that the COCAP obtains the highest overall hit ratio and CO3S obtains pretty close overall hit ratio to COCAP. This is due to higher edge and core router hit ratio by COCAP compared to CO3S depicted in Figure 6.18 and Figure 6.19. Comparing CO3S with LCD, we find that CO3S obtains less edge hit ratio than LCD depicted in Figure 6.18 but it compensates through getting higher hit ratio at core routers depicted in Figure 6.19.

Regarding comparison between CO2S and LCE (universal LRU caching) at the edge router, Figure 6.18 shows that CO2S obtains higher hit ratio compared to LRU. However, we show in Section 3.1.1 that LRU and two-state policy obtain similar hit ratios for standalone cache that is the case at the edge router because there is no cross traffic and edge routers do not get affected by other routers in the binary topology. The difference is due to the effect of stream of packets. That is, there is a kind of thrashing problem because in this experiment the content size is 500 packets on average. For example, suppose that the most popular content has 500 packets. Then, the edge router with cache sizes smaller than 500 does not have enough slots to store the most popular content. This leads to the situation that even the most popular packets evict themselves from the cache. This problem is intensified by having multiple active flows because the packets of one flow evict

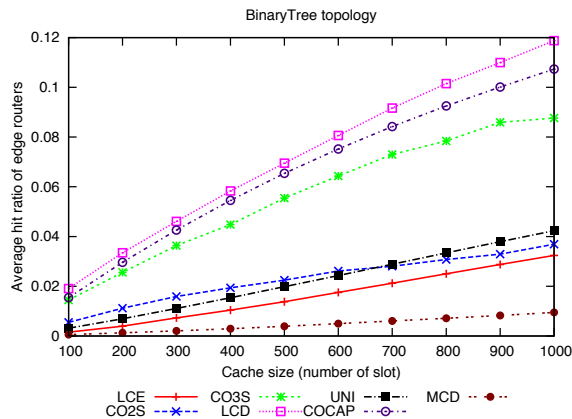


Figure 6.18: Average hit ratio of edge routers

the packets of another flow. This thrashing problem affects the performance of the LRU but not of two-state policy due to freezing. Figure 6.18 also shows that the hit ratio of UNI is also greater than LRU on the edge routers because UNI only writes some of the packets to the edge router. This decreases the effect of thrashing on the edge router.

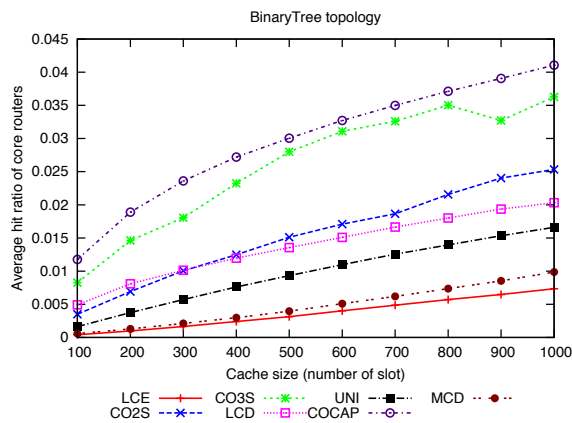


Figure 6.19: Average hit ratio of core routers

2) Average content download time: Figure 6.20 shows that the COCAP decreases the content download time up to 17% more than other schemes because COCAP obtains the high hit ratio at both core and edge routers.

3) Average eviction rate per slot: The eviction rate is an important factor for an ICN router. The less the eviction rate is, the less memory write happens in an

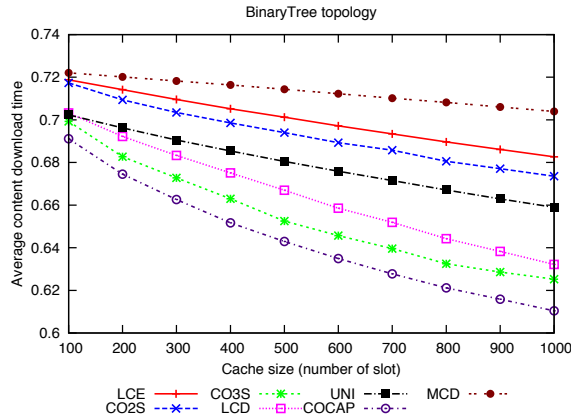


Figure 6.20: Average content download time

ICN router that should operate at line speed [6]. Moreover, the less eviction rate

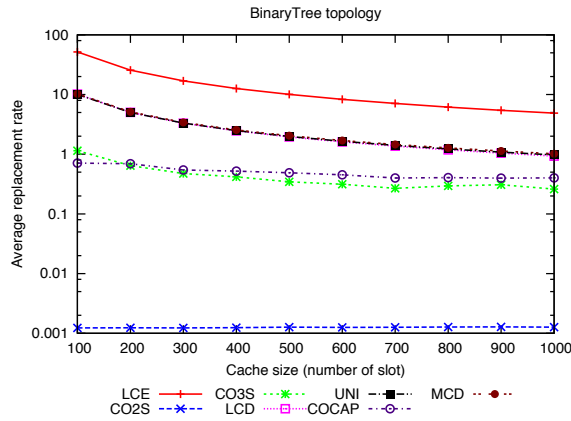


Figure 6.21: Average eviction rate per slot

can lead to reducing the energy consumption in ICN routers. Figure 6.21 shows that CO2S has less average eviction rate up to four orders of magnitude compared to other schemes because CO2S transits to the frozen state faster than CO3S and COCAP. In addition, CO2S only evicts the contents at the updating state. The highest eviction rate belongs to LCE because LCE writes a missed content to all of the routers in the path.

Performance versus Content Size

Setting: In this experiment, we use the constant cache size of 300 packets and

increase the average content size from 100 packets to 600 packets. Other setting is the same as the previous experiment.

Findings and Discussion: Figure 6.22, 6.23, 6.24, 6.25, and 6.26 show that the performance metrics have the similar trend as the previous section. Therefore, we skip the discussion in this section.

1) Overall hit ratio:

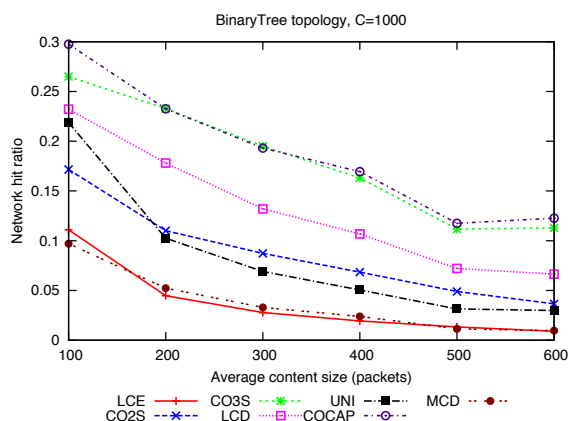


Figure 6.22: Overall network hit ratio

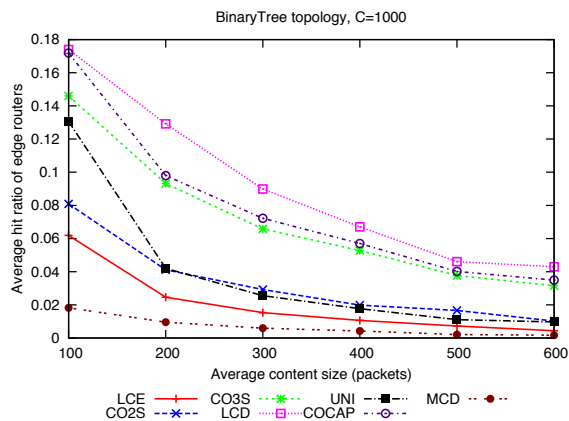


Figure 6.23: Average hit ratio of edge routers

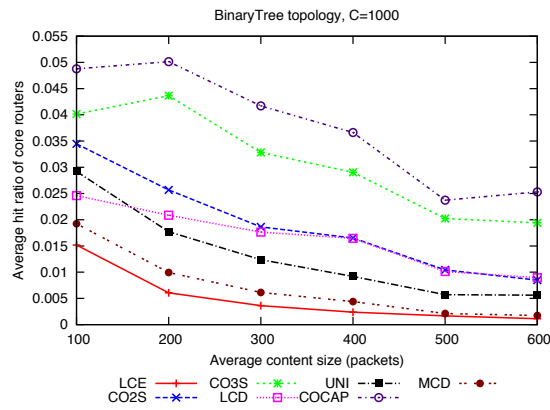


Figure 6.24: Average hit ratio of core routers

2) Average content download time:

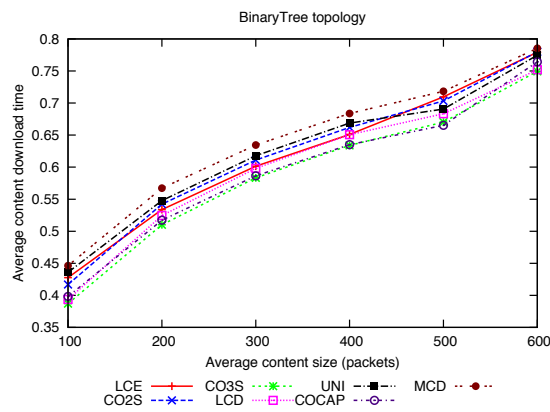


Figure 6.25: Average content download time

3) Average eviction rate per slot:

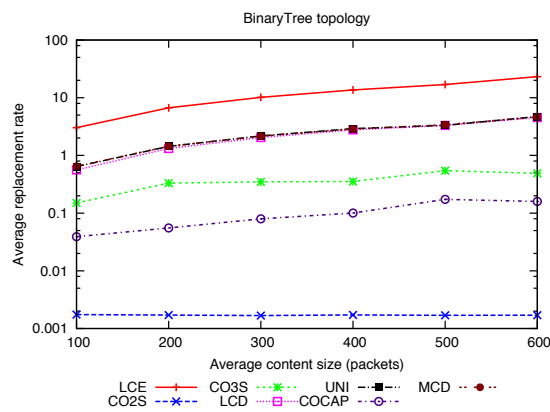


Figure 6.26: Average eviction rate per slot

6.2.5 Real ISP Topology Capture by RocketFuel

Setting: We use the topology depicted in Figure 6.27 with ISP number of 6461 for our last evaluation. The ISP has 25 edge routers and 77 core routers. There are 5×10^5 contents in the network and the producers of these contents are connected to 25 edge routers (green) and 25 core routers (blue). In addition, there are three different types of traffic: i) Internet video ii) Web iii) file sharing. We select the content size of these traffic types such that the fraction of video, web and file sharing traffic are 64%, 21% and 15% respectively due to the prediction about the future traffic pattern [22]. The Internet video traffic has the average content size of 600 packets; the web traffic has the average content size of 100 packets; and the file sharing has the average content size of 2000 packets. The content size (number of packets per content) is generated based on the geometric distribution [33]. Consequently, there are around 2×10^7 different packets in the network. The

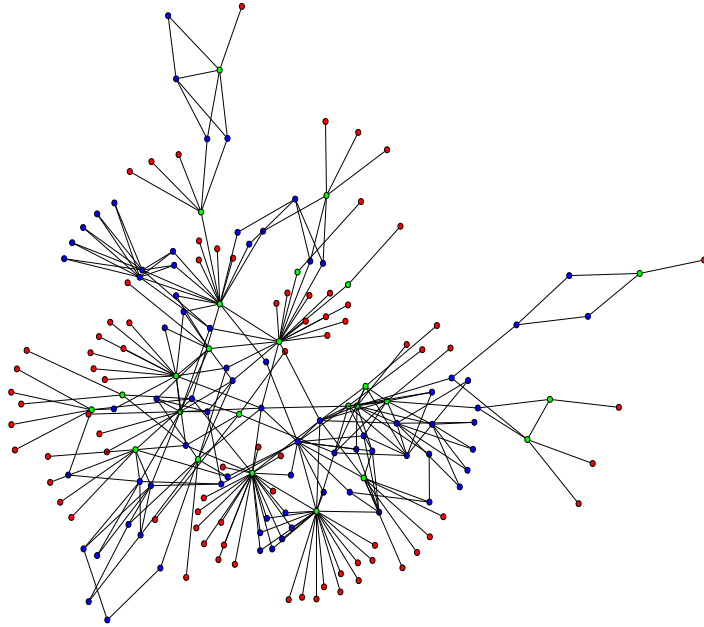


Figure 6.27: Topology captured by rocketfuel with ISP number of 6461

consumers generate requests with the average rate of 250 requests per second. The popularity of the content follows the Zipf distribution with a slope of one ($\alpha = 1$).

The popularity changes with the method described in Section 3.3 with $p_s = 0.05$ and $\lambda_c = 0.00066$ (period of 1500 seconds). We run the simulation for 6000 seconds and the updating period of CO2S, CO3S and COCAP are 700 seconds. We do the experiment for the virtual cache size of 4000 and 8000 packets and obtain the same trend.

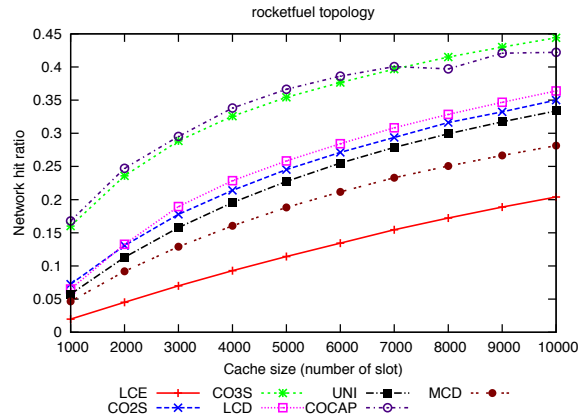


Figure 6.28: Overall network hit ratio

Findings and Discussion:

1) **Overall hit ratio:** Figure 6.28 shows that the COCAP and CO3S have the highest overall hit ratio among all of the schemes. Figure 6.29 shows that COCAP and CO3S obtain the higher hit ratios than LCD at the edge router because COCAP and CO3S prioritize the routers based on their closeness ranks with respect to different paths. However, LCD writes all of the contents, missed in the network, to the edge routers because they are connected to producers. This decreases the hit ratio of edge routers.

Figure 6.30 shows that the COCAP and CO3S have the highest hit ratio for the core routers for small cache sizes. Although, increasing the cache size increases their core hit ratios, they get downgraded to lower ranks by increasing the cache size because of the updating period effect. That is, increasing the cache size increases the time required for a cache to reach frozen state. Therefore, after a certain point (5000 packets in Figure 6.30), the updating period of 700 seconds is not sufficient for both core and edge routers to reach frozen state. Consequently,

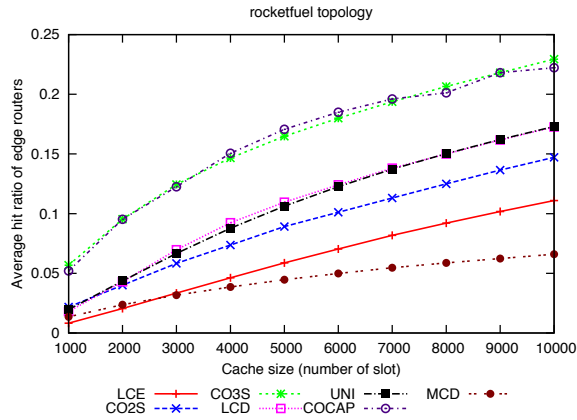


Figure 6.29: Average hit ratio of edge routers

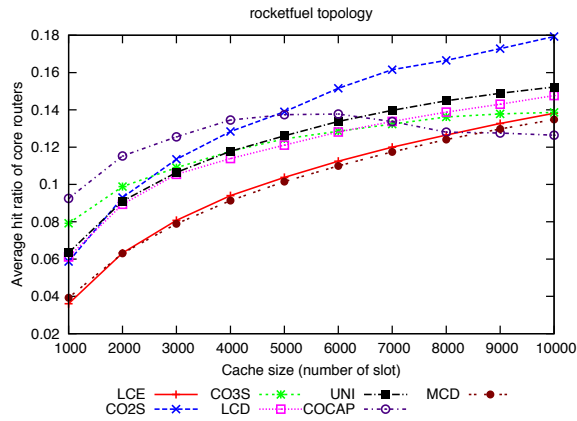


Figure 6.30: Average hit ratio of core routers

core routers keep replacing the packets and affect the requests of each other such that they cannot obtain high hit ratio. As it is shown in the figure, the updating period affects COCAP more than CO3S at core routers because the condition for a content to be protected by COCAP is harder than a content to be frozen by CO3S (described in Section 6.2.2). Consequently, the time required for COCAP to make the whole cache slots protected is longer than CO3S. However, the role of edge router for large cache sizes is more important than core routers as Figure 6.30 and 6.29 show that both COCAP and CO3S obtain similar overall and edge router hit ratios.

2) Average content download time: Figure 6.31 shows that the worst average download time belongs to MCD because MCD removes the hit contents from the

routers after getting a hit to decrease the number of duplicate copies. Therefore, when a miss happens at the edge router, the content is fetched from producer with high probability.

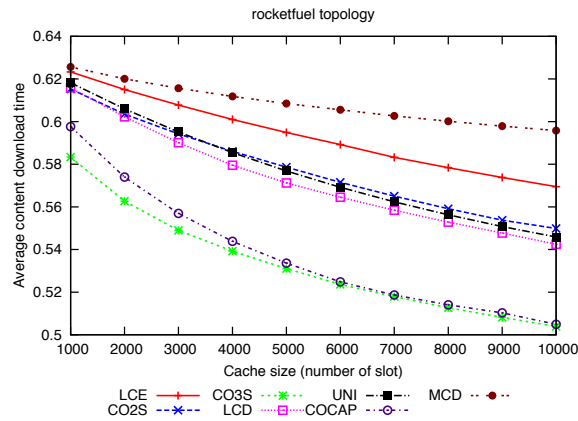


Figure 6.31: Average content download time

3) Transferred packet reduction ratio: Figure 6.32 shows that the CO3S and COCAP obtain larger transferred reduction ratio up to 28% compared to other schemes.

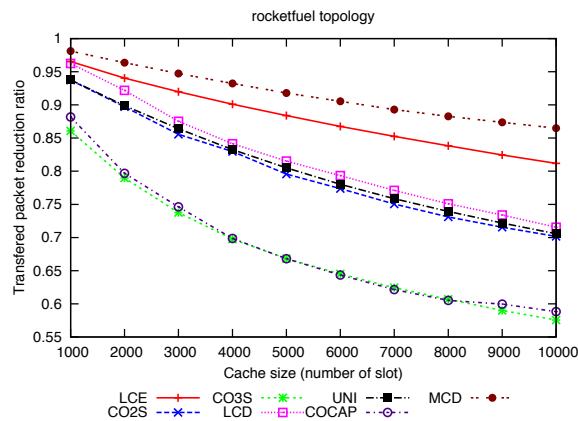


Figure 6.32: Transferred packet reduction ratio

4) Average eviction rate per slot:

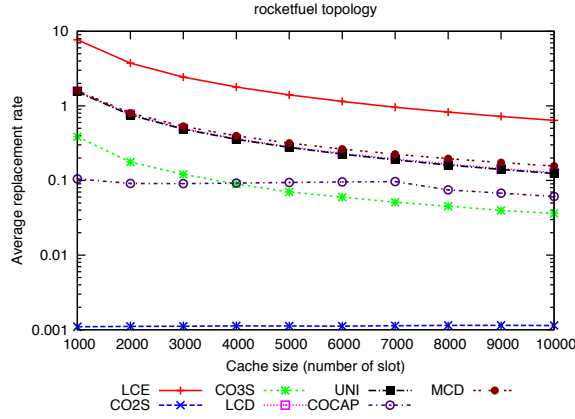


Figure 6.33: Average eviction rate per slot

6.3 Summary

In this chapter, we complete our second approach in proposing the lightweight coordinated schemes by proposing COCAP. The COCAP uses the concept of virtual cache to bring the popular contents close to the consumers with respect to the path. We also evaluate all of our proposed coordinated schemes (CO2S, CO3S and COCAP) for synthetic and real topologies. Our evaluation shows that COCAP and CO3S obtain the best performance among all of the applicable schemes for ICN network of caches and they are followed by LCD and CO2S that obtain similar performance. Moreover, the COCAP outperforms the CO2S and CO3S for topologies without cross traffic. This is due to obtaining a higher hit ratio at the edge routers compared to CO2S and CO3S. Moreover, COCAP obtains very close overall hit ratio to the CO3S for ISP topology with cross traffic. However, CO3S performs better than COCAP in term of average content download time and traffic reduction ratio. Finally, our evaluations show that the CO2S obtains the comparable results with LCD but CO2S reduces the average eviction rate per slot up to four orders of magnitude.

Chapter 7

Conclusion

We conclude the thesis by a summary of our contributions followed by discussion of future work.

7.1 Thesis Summary

Recently, ICN is introduced to make the Internet communication model consistent with the usage of Internet. Although there are different ICN proposals, all of them propose to integrate the router and cache because current and future Internet traffic are cachable. As a result, we have a network of caches in the scale of Internet that requires lightweight coordinated schemes. These lightweight schemes should address the filtering effect problem and manage the redundant copies of a content to obtain a high overall hit ratio. This thesis presents the coordinated caching schemes with low overhead that can solve the filter effect problem, manage the redundancy and obtain a high overall hit ratio. We propose two coordinated schemes: n-state and COCAP.

Our first coordinated caching scheme, n-state is based on our new policy called two-state. The two-state policy introduces a new type of filtering effect. Due to this new filtering property, using the two-state policy at the edge router leads to providing opportunity for the core routers to obtain high hit ratios. Further-

more, our two-state policy and the LRU replacement policy have the same hit ratios under the Independent Reference Model (IRM) assumption. The two-state has a reservation mechanism that improves its adaptability to the traffic pattern changes in the network with large RTTs. Although two-state suffers from pollution problem, its generalization (n -state), overcomes the pollution problem for $n \geq 3$.

The n -state policy obtains higher hit ratio than two-state by capturing the popular contents and solving the pollution problem. The improvement of cache hit ratio is considerable by increasing the n from two to three but the achievement is negligible for increasing the $n > 3$. Under IRM assumption, the three-state policy obtains the hit ratio close to the hit ratio of LFU. Moreover, using trace-based simulation, we show that the three-state policy obtains a high hit ratio for a standalone cache and provides the opportunity for subsequent caches to obtain a high hit ratio. Although n -state provides the opportunity for other caches, obtaining a high overall hit ratio without coordination is almost impossible because of the redundant copies of the contents.

Our first coordinated scheme integrated with n -state policy obtains the advantages of n -state policy and manages the redundant copies. Our scheme has the property of removing the useless redundant copies and prioritizing the routers based on their closeness rank with respect to different paths. This leads to cache the popular contents close to the consumers with respect to paths. We present two versions of our coordinated scheme, CO3S and CO2S. Compared to other work, the CO2S decreases the eviction rate up to four orders of magnitude while it obtains comparable performance in terms of the overall hit ratio, content download time and the transferred packet. Moreover, our CO3S, outperforming CO2S, improves the overall hit ratio up to seven times for small cache sizes and up to 25% for large cache sizes compared to LCE. Consequently, CO3S reduces the content download time 24% and the transferred packet 7% to 13% more than LCE. The implementation of our coordinated scheme is simple in terms of processing and communicating overhead.

Our second coordinated scheme is based on the CAP policy that is designed to address all of the problems for a standalone cache. CAP addresses the pollution problem by dividing the cache into two variable sized segments: protected and unprotected. The missed contents are written into unprotected segment and they are moved into protected segment if they get at least one hit before being evicted. Therefore, the one-timer contents do not affect the popular content in the protected section. We assign one independent policy for each segment. Based on the advantages and disadvantages of different replacement policy combinations, we choose the RND for the unprotected segment and do nothing (no action for a content hit) for the protected segment. This combination can overcome the contention problem and at the same time it is resistant against the thrashing problem. Finally, the time complexity of CAP is constant and it does not impose memory overhead. Our evaluation through simulation of both synthetic and real workloads shows that CAP obtains the performance close to the state-of-the-art policy in terms of cache hit ratio.

We propose our second coordinated scheme based on CAP policy, COCAP, that introduces the concept of virtual cache. The virtual cache enables the COCAP to cache the popular content close to the consumers with respect to the path. We evaluate COCAP for synthetic and real topology captured by rocket-fuel. For topologies without cross traffic, the COCAP outperforms the CO2S and CO3S. This is due to obtaining a higher hit ratio at the edge routers by COCAP. Moreover, COCAP obtains very close overall hit ratio to the CO3S for ISP topology with cross traffic. However, CO3S performs better than COCAP in term of average content download time and traffic reduction ratio.

7.2 Future Work

7.2.1 The Effect of Coordinated Schemes on Routing

Our coordinated schemes obtain high hit ratios at both edge and core routers. Therefore, the cache size of the core routers and the number of core routers involved in a path affect the cache hit ratio. The number of routers involved in a path depends on network topology and routing algorithm. Therefore, the topology and routing protocols causing larger number of routers in a path lead to the higher overall network hit ratio. The higher overall hit ratio leads to saving bandwidth by avoiding redundant transmission and decreasing the access delay by bringing popular contents close to the consumers. On the other hand, increasing the number of routers in a path increases the bandwidth consumptions and access delay. Therefore, there is a trade-off that can be a valuable research direction.

7.2.2 Combining CO2S, CO3S and COCAP with Traffic Engineering

The objective of our coordinated schemes is to obtain a high overall hit ratio. However, a specific strategy can integrate with our scheme to obtain a specific goal. For example, an ISP may be interested to minimize the traffic coming from a list of ISPs. This can be integrated into our schemes by considering the traffic engineering metrics. Another example is the number of hops between a cache and the producer that can be used to value a content. The farther a packet from its producer, the more important the packet can be considered. Finally, the number of hops between a cache and the consumer of the content packet can also be considered.

7.2.3 Enhancing the Coordinated Schemes by Considering Neighbour Caches

Our coordinated schemes use n-state and CAP to cache a number of contents and do not replace them for a while. Consequently, as we showed, the eviction rate is drastically decreased compared to other schemes. This provides an opportunity for combining our schemes with the work that considers the neighbour caches such as Summary Cache. These schemes change the default path of a request by considering the state of neighbour caches and have high communication overhead where the replacement policy is used because each cache should update its neighbours about its state changes with high rate. However, our schemes can be combined with these schemes without imposing a high communication overhead because the update rate, affected by the updating period, is lower.

Bibliography

- [1] B. Ahlgren, M. D'Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone. Design considerations for a network of information. In *Proceedings of the 2008 ACM CoNEXT Conference*, pages 1–6. ACM, 2008.
- [2] J. Almeida and a.Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, June 2000.
- [3] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee. Redundancy in network traffic: findings and implications. In *SIGMETRICS '09*, pages 37–48, New York, New York, USA, 2009. ACM Press.
- [4] I. Ari, A. Amer, and R. Gramacy. ACME: adaptive caching using multiple experts. In *WDAS'02*, volume 14, pages 143–158, 2002.
- [5] S. Arianfar, P. Nikander, and J. Ott. On content-centric router design and implications. In *Proceedings of the Re-Architecting the Internet Workshop on - ReARCH '10*, volume 9, page 6, New York, New York, USA, 2010. ACM Press.
- [6] S. Arianfar, P. Nikander, and J. Ott. Packet-level caching for information-centric networking. *Finnish ICT-SHOK Future Internet Project, Tech. Rep*, 2010.

BIBLIOGRAPHY

- [7] H. Bahn, K. Koh, S. Noh, and S. Lyul. Efficient replacement of nonuniform objects in web caches. *Computer*, (June):65–73, 2002.
- [8] G. Bai and C. Williamson. Time-domain analysis of Web cache filter effects. *Performance Evaluation*, 58(2-3):285–317, Nov. 2004.
- [9] S. Bansal and D. Modha. CAR: Clock with adaptive replacement. In *FAST '04 Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pages 187–200, 2004.
- [10] J. V. D. Berg. LRU is better than FIFO under the independent reference model. *Journal of applied probability*, 29(1):239–243, 1992.
- [11] S. Bhattacharjee, K. Calvert, and E. Zegura. Self-organizing wide-area network caches. In *Proceedings. IEEE INFOCOM '98*, volume 2, pages 600–608. IEEE, 1998.
- [12] S. Borst, V. Gupta, and A. Walid. Distributed caching algorithms for content distribution networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.
- [13] A. Brodnik, S. Carlsson, and E. Demaine. Resizable arrays in optimal time and space. *Algorithms and Data Structures Lecture Notes in Computer Science*, 1663:37–48, 1999.
- [14] M. Cha, H. Kwak, and P. Rodriguez. Analyzing the video popularity characteristics of large-scale user generated content systems. *IEEE/ACM Transactions on Networking*, 17(5):1357–1370, Oct. 2009.
- [15] W. Chai, D. He, I. Psaras, and G. Pavlou. Cache Less for More in Information-Centric Networks. In *IFIP NETWORKING 2012*, volume 2012, pages 27–40, 2012.

BIBLIOGRAPHY

- [16] H. Che, Y. Tung, and Z. Wang. Hierarchical Web caching systems: modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, 20(7):1305–1314, Sept. 2002.
- [17] X. Chen, Q. Fan, and H. Yin. Caching in Information-Centric Networking: From a content delivery path perspective. In *2013 9th International Conference on Innovations in Information Technology (IIT)*, pages 48–53. IEEE, Mar. 2013.
- [18] E. Chlebus and J. Brazier. Nonstationary poisson modeling of web browsing session arrivals. *Information Processing Letters*, 102(5):187–190, 2007.
- [19] K. Cho, M. Lee, K. Park, T. T. Kwon, and Y. Choi. WAVE: Popularity-based and collaborative in-network caching for content-oriented networks. *2012 Proceedings IEEE INFOCOM Workshops*, pages 316–321, Mar. 2012.
- [20] H.-g. Choi, J. Yoo, T. Chung, N. Choi, T. Kwon, and Y. Choi. CoRC. In *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems - ANCS '14*, pages 161–172, New York, New York, USA, 2014. ACM Press.
- [21] J. Choi, A. S. Reaz, and B. Mukherjee. A Survey of User Behavior in VoD Service and Bandwidth-Saving Multicast Streaming Schemes. *IEEE Communications Surveys and Tutorials*, 14(1):156–169, Jan. 2012.
- [22] Cisco Visual Networking Index. Cisco Visual Networking Index : Forecast and Methodology, 2011-2016. Technical report.
- [23] J. E. G. Coffman and P. J. Denning. *Operating Systems Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1973.
- [24] F. Corbato. A paging experiment with the multics system. *MIT Press*, Honor of P:217–228, 1968.

BIBLIOGRAPHY

- [25] P. B. Danzig, R. S. Hall, and M. F. Schwartz. A case for caching file objects inside internetworks. In *SIGCOMM '93*, pages 239–248, New York, New York, USA, 1993. ACM Press.
- [26] X. Ding, S. Jiang, and X. Zhang. BP-Wrapper: A System Framework Making Any Replacement Algorithms (Almost) Lock Contention Free. *2009 IEEE 25th International Conference on Data Engineering*, pages 369–380, Mar. 2009.
- [27] L. Dong, D. Zhang, Y. Zhang, and D. Raychaudhuri. Optimal Caching with Content Broadcast in Cache-and-Forward Networks. *2011 IEEE International Conference on Communications (ICC)*, pages 1–5, June 2011.
- [28] L. Dong, D. Zhang, Y. Zhang, and D. Raychaudhuri. Performance evaluation of content based routing with in-network caching. *2011 20th Annual Wireless and Optical Communications Conference (WOCC)*, pages 1–6, Apr. 2011.
- [29] R. P. Doyle, J. S. Chase, S. Gadde, and A. M. Vahdat. The Trickle-Down Effect: Web Caching and Server Request Distribution. *Computer Communications*, 25(4):345–356, Mar. 2002.
- [30] S. Eum, K. Nakauchi, T. Usui, M. Murata, and N. Nishinaga. Potential based routing for ICN. *Proceedings of the 7th Asian Internet Engineering Conference on - AINTEC '11*, pages 116–119, 2011.
- [31] R. Fonseca and V. Almeida. On the intrinsic locality properties of web reference streams. *INFOCOM 2003*, 00(C), 2003.
- [32] C. Fricker, P. Robert, J. Roberts, and N. Sbihi. Impact of traffic mix on caching performance in a content-centric network. *2012 Proceedings IEEE INFOCOM Workshops*, pages 310–315, Mar. 2012.
- [33] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. Youtube traffic characterization a view from the edge. In *Proceedings of the 7th ACM SIGCOMM conference*

BIBLIOGRAPHY

- on Internet measurement - IMC '07*, page 15, New York, New York, USA, 2007. ACM Press.
- [34] G. Glass and P. Cao. Adaptive page replacement based on memory reference behavior. *ACM SIGMETRICS Performance Evaluation Review*, 25(1):115–126, June 1997.
- [35] H. Gomaa, G. G. Messier, C. Williamson, and R. Davies. Estimating Instantaneous Cache Hit Ratio Using Markov Chain Analysis. *IEEE/ACM Transactions on Networking*, 21(5):1472–1483, Oct. 2013.
- [36] M. Gritter and D. Cheriton. An architecture for content routing support in the internet. In *Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems-Volume 3*, pages 4–15. USENIX Association, 2001.
- [37] S. Guo. Collaborative forwarding and caching in content centric networks. *Espringer NETWORKING 2012*, 7289/2012:41–55, 2012.
- [38] A. Ioannou and S. Weber. Towards on-path caching alternatives in Information-Centric Networks. In *39th Annual IEEE Conference on Local Computer Networks*, pages 362–365. Ieee, Sept. 2014.
- [39] IRCache. Internet traffic archive, <ftp://ftp.ircache.net/Traces/DITL-2007-01-09/>.
- [40] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.
- [41] S. Jiang, F. Chen, and X. Zhang. CLOCK-Pro: an effective improvement of the CLOCK replacement. In *ATEC '05 Proceedings of the annual conference on USENIX Annual Technical Conference*, 2005.

BIBLIOGRAPHY

- [42] S. Jiang and X. Zhang. LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Performance Evaluation Review*, (i), 2002.
- [43] S. Jiang and X. Zhang. Making LRU friendly to weak locality workloads: A novel replacement algorithm to improve buffer cache performance. *Computers, IEEE Transactions on*, 54(8):939–952, 2005.
- [44] S. Jin and A. Bestavros. Sources and characteristics of Web temporal locality. In *Proceedings of the 8th MASCOTS*, pages 28–35. IEEE Comput. Soc, 2000.
- [45] R. Karedla, J. Love, and B. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, Mar. 1994.
- [46] J. Kim, J. Choi, J. Kim, S. Noh, and S. Min. A low-overhead high-performance unified buffer management scheme that exploits sequential and looping references. *OSDI 00*, 1, 2000.
- [47] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. *ACM SIGCOMM Computer Communication Review*, 37(4):181, Oct. 2007.
- [48] N. Laoutaris and H. Che. The LCD interconnection of LRU caches and its analysis. *Performance Evaluation*, pages 1–33, 2006.
- [49] N. Laoutaris, S. Syntila, and I. Stavrakakis. Meta algorithms for hierarchical Web caches. In *IEEE International Conference on Performance, Computing, and Communications, 2004*, pages 445–452. IEEE, 2004.
- [50] D. Lee, J. Choi, J.-H. Kim, S. Noh, S. L. Min, Y. Cho, and C. S. Kim. LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12):1352–1361, Dec. 2001.

BIBLIOGRAPHY

- [51] J. Li, B. Liu, and H. Wu. Energy-Efficient In-Network Caching for Content-Centric Networking. *IEEE Communications Letters*, 17(4):797–800, Apr. 2013.
- [52] K. Li, H. Shen, F. Y. L. Chin, and S. Q. Zheng. Optimal methods for coordinated enroute web caching for tree networks. *ACM Transactions on Internet Technology*, 5(3):480–507, Aug. 2005.
- [53] W. Li, E. Chan, G. Feng, D. Chen, and S. Lu. Analysis and performance study for coordinated hierarchical cache placement strategies. *Computer Communications*, 33(15):1834–1842, Sept. 2010.
- [54] H. Madhyastha, E. Katz-Bassett, and T. Anderson. iPlane Nano: path prediction for peer-to-peer applications. *NSDI*, (April):137–152, 2009.
- [55] A. Mahanti and C. Williamson. Traffic analysis of a web proxy caching hierarchy. *Network, IEEE*, (June):16–23, 2000.
- [56] R. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.
- [57] N. Megiddo and D. Modha. ARC: A self-tuning, low overhead replacement cache. In *FAST '03 Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 115–130, 2003.
- [58] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive web caching: towards a new global caching architecture. *Computer Networks and ISDN Systems*, 30(22-23):2169–2177, Nov. 1998.
- [59] Z. Ming, M. Xu, and D. Wang. Age-based cooperative caching in Information-Centric Networks. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 268–273. IEEE, Mar. 2012.

BIBLIOGRAPHY

- [60] S. Mitra, M. Agrawal, A. Yadav, N. Carlsson, D. Eager, and A. Mahanti. Characterizing Web-Based Video Sharing Workloads. *ACM Transactions on the Web*, 5(2):1–27, 2011.
- [61] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical Power Management for Enterprise Storage. *ACM Transactions on Storage*, 4(3):1–23, Nov. 2008.
- [62] V. F. Nicola, A. Dan, and D. M. Dias. Analysis of the generalized clock buffer replacement scheme for database transaction processing. In *Proceedings of the 1992 ACM SIGMETRICS - SIGMETRICS '92/PERFORMANCE '92*, volume 20, pages 35–46, New York, New York, USA, 1992. ACM Press.
- [63] NTP. Network Time Protocol, <http://www.ntp.org/ntpfaq/NTP-s-def.htm>.
- [64] E. J. O’Neil, P. E. O’Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93*, volume 00, pages 297–306, New York, New York, USA, 1993. ACM Press.
- [65] V. Paxson. End-to-end routing behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, 1997.
- [66] D. Povey and J. Harrison. A distributed Internet cache. *Australian Computer Science Communications*, 1997.
- [67] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic In-Network Caching for Information-Centric Networks. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, pages 1–6. ACM, 2012.
- [68] J. Ren, W. Qi, C. Westphal, J. Wang, K. Lu, S. Liu, and S. Wang. MAGIC : a Distributed MAX-Gain In-network Caching Strategy in Information-Centric Networks. pages 470–475, 2014.
- [69] RFC 1305. Network Time Protocol, 1992.

BIBLIOGRAPHY

- [70] J. Robinson. Data cache management using frequency based replacement. *Computing*, pages 134–142, 1990.
- [71] P. Rodriguez, C. Spanner, and E. Biersack. Web caching architectures: hierarchical and distributed caching. In *Proceedings of WCW*, volume 99, pages 1–15. Citeseer, 1999.
- [72] E. Rosensweig and J. Kurose. Breadcrumbs: efficient, best-effort content location in cache networks. In *INFOCOM 2009, IEEE*, pages 2631–2635. IEEE, 2009.
- [73] D. Rossi and G. Rossini. On sizing CCN content stores by exploiting topological information. *IEEE NOMEN Workshop*, 2012.
- [74] G. Rossini and D. Rossi. A dive into the caching performance of Content Centric Networking. In *IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks*, 2012.
- [75] S. Saha, A. Lukyanenko, and A. Yla-Jaaski. Cooperative caching through routing control in information-centric networks. In *2013 Proceedings IEEE INFOCOM*, pages 100–104. IEEE, Apr. 2013.
- [76] D. Shasha and T. Johnson. 2Q: A low overhead high performance buffer management replacement algorithm. In *VLDB*, pages 439–450, 1994.
- [77] Y. Smaragdakis, S. Kaplan, and P. Wilson. EELRU: Simple and Effective Adaptive Page Replacement. *ACM SIGMETRICS Performance Evaluation Review*, 27(1):122–133, June 1999.
- [78] A. J. Smith. Sequentiality and prefetching in database systems. *ACM Transactions on Database Systems*, 3(3):223–247, Sept. 1978.
- [79] V. Sourlas, P. Flegkas, L. Gkatzikis, and L. Tassiulas. Autonomic cache management in Information-Centric Networks. *2012 IEEE Network Operations and Management Symposium*, pages 121–129, Apr. 2012.

BIBLIOGRAPHY

- [80] V. Sourlas, L. Gkatzikis, P. Flegkas, and L. Tassiulas. Distributed Cache Management in Information-Centric Networks. *IEEE Transactions on Network and Service Management*, 10(3):286–299, Sept. 2013.
- [81] D. Starobinski and D. Tse. Probabilistic methods for web caching. *Performance Evaluation*, 46(2-3):125–137, Oct. 2001.
- [82] X. Tang and S. Chanson. Coordinated en-route web caching. *Computers, IEEE Transactions on*, 51(6):595–607, 2002.
- [83] I. Tatarinov, A. Rousskov, and V. Soloviev. Static caching in Web servers. *Proceedings of Sixth International Conference on Computer Communications and Networks*, pages 410–417, 1997.
- [84] G. Tyson, S. Kaune, S. Miles, Y. El-khatib, A. Mauthe, and A. Taweel. A Trace-Driven Analysis of Caching in Content-Centric Networks. In *ICCCN*, pages 1–7, 2012.
- [85] B. Venkataraman, R. L. Shamanna, and I. Rhee. Advertising cached contents in the control plane: Necessity and feasibility. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 286–291. IEEE, Mar. 2012.
- [86] J. M. Wang and B. Bensaou. Progressive caching in CCN. In *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 2727–2732. IEEE, Dec. 2012.
- [87] D. Weikle, S. McKee, and W. Wulf. Caches as filters: A new approach to cache analysis. In *MASCOTS'98*, volume 2, pages 2–12, Montreal, Feb. 1998.
- [88] C. Williamson. On filter effects in web caching hierarchies. *ACM Transactions on Internet Technology*, 2(1):47–77, Feb. 2002.
- [89] T. Wong, G. Ganger, and J. Wilkes. My cache or yours? making storage more exclusive. *USENIX Annual Technical Conference*, (June):161–175, 2002.

BIBLIOGRAPHY

- [90] H. Xie, G. Shi, and W. Pengwei. TECC: Towards collaborative in-network caching guided by traffic engineering. In *2012 Proceedings IEEE INFOCOM*, pages 2546–2550. IEEE, Mar. 2012.
- [91] M. Xie, I. Widjaja, and H. Wang. Enhancing cache robustness for content-centric networking. *INFOCOM, 2012 Proceedings IEEE*, pages 2426–2434, 2012.
- [92] J. Yang, W. Wang, and R. Muntz. Collaborative Web caching based on proxy affinities. *ACM SIGMETRICS Performance Evaluation Review*, 28(1):78–89, June 2000.
- [93] Y. Zhang, G. Soundararajan, M. W. Storer, and L. N. Bairavasundaram. Warming up Storage-Level Caches with Bonfire. In *FAST 13*, 2013.
- [94] Y. Zhou, J. Philbin, and K. Li. The multi-queue replacement algorithm for second level buffer caches. In *USENIX Annual Tech*, 2001.

Appendix A

Synthetic Workload for Two-State Policy

The objective of this appendix is to show that our two properties of two-state policy introduced and evaluated in Chapter 3 are valid under IRM assumption for a wide range of contents number (N), cache sizes (C) and zipf slopes (α). The first property is that using two-state policy at the edge routers provides better opportunity for core routers compared to the situation that replacement policies (RND, FIFO and LRU) manage the edge router. The second property is that two-state can obtain the hit ratio same as LRU. We only present the results in the appendixes because we discussed the reasons for these properties in main chapters.

To evaluate the properties under different combinations of the N , C and α , we select $N = 100$, $N = 1000$ and $N = 10000$ and plot the stack distance metrics, second and first cache hit ratios while either C or α is constant and the other one varies. In addition, we stick to the same topology used in Section 3.1.2. Firstly, we plot the curves versus C for each N and $0.6 \leq \alpha \leq 1.5$ with step of 0.1. However, we present the results only for minimum (0.6), average (1) and maximum (1.5) values of α due to the similar trends. Secondly, we plot the curves versus α for each N and $10 \leq C \leq N$. However, we present the results only for three small cache sizes compared to the N because our concentration is on ICN network of

APPENDIX A. SYNTHETIC WORKLOAD FOR TWO-STATE POLICY

caches that has a small $\frac{C}{N}$. In the next appendix, we evaluate the two above described properties using trace-based simulations.

A.1 Content Number of 100

A.1.1 Stack Distance

Average

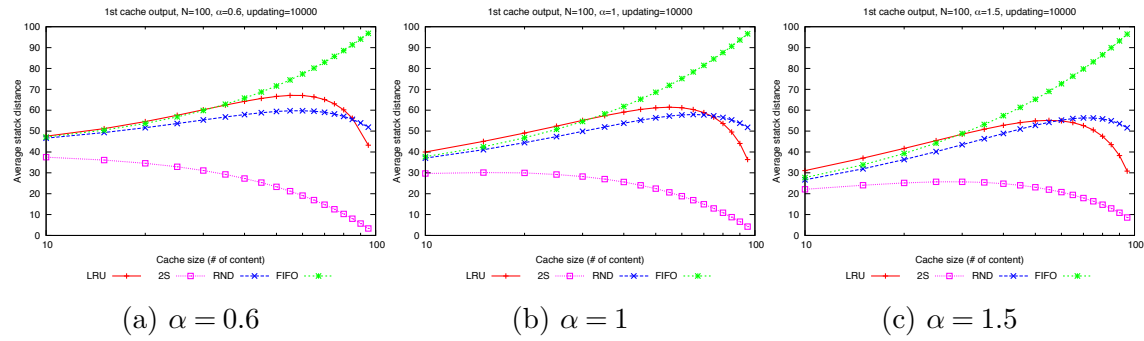


Figure A.1: Average stack distance with different α , $N=100$

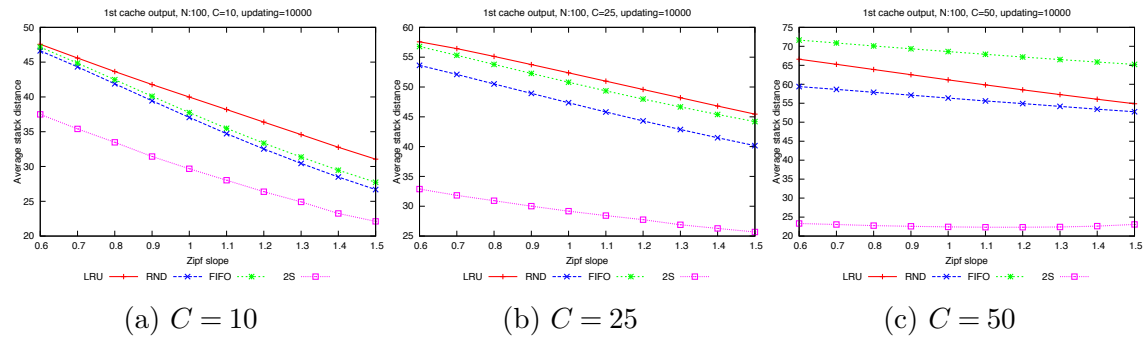


Figure A.2: Average stack distance with different cache sizes, $N=100$

APPENDIX A. SYNTHETIC WORKLOAD FOR TWO-STATE POLICY

Minimum

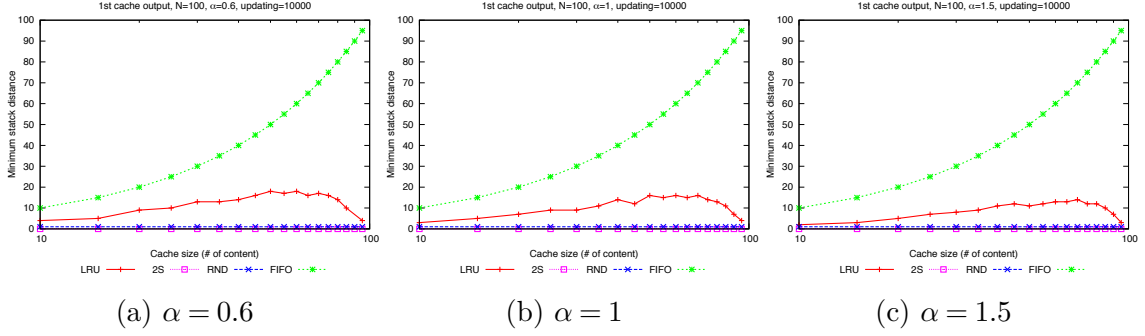


Figure A.3: Minimum stack distance with different α , $N=100$

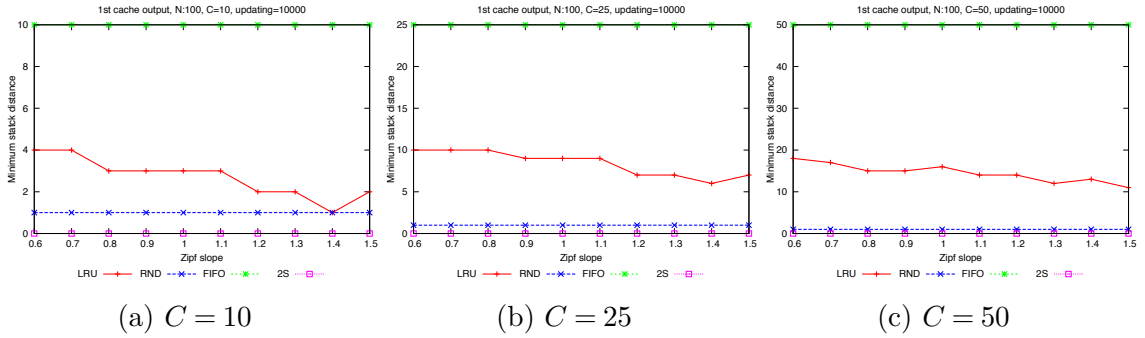


Figure A.4: Minimum stack distance with different cache sizes, $N=100$

A.1.2 The Effect on Overall Cache Hit Ratio

FIFO on Second Cache

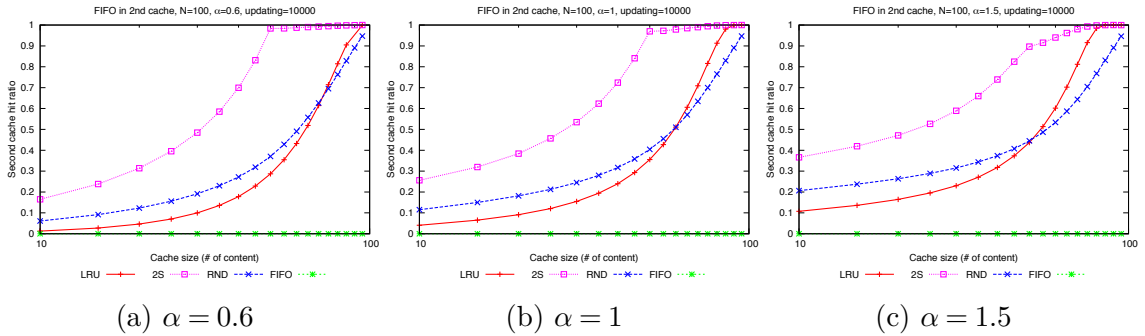


Figure A.5: Second cache (FIFO) hit ratio with different α , $N=100$

APPENDIX A. SYNTHETIC WORKLOAD FOR TWO-STATE POLICY

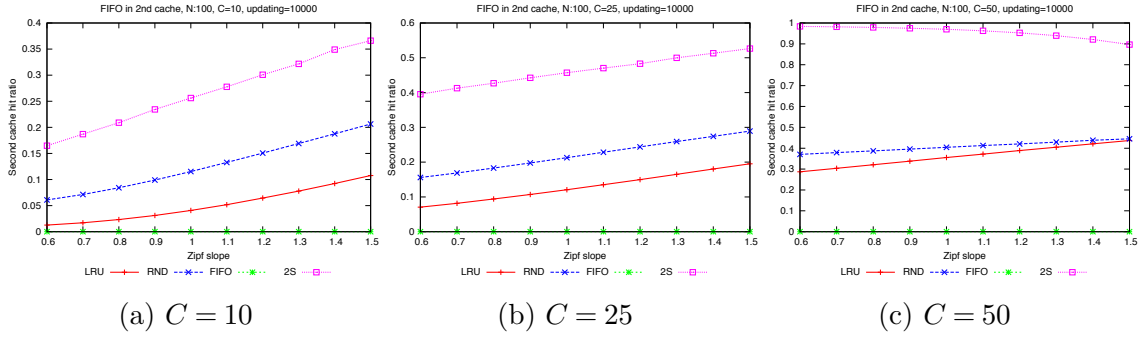


Figure A.6: Second cache (FIFO) hit ratio with different cache sizes, $N=100$

LRU on Second Cache

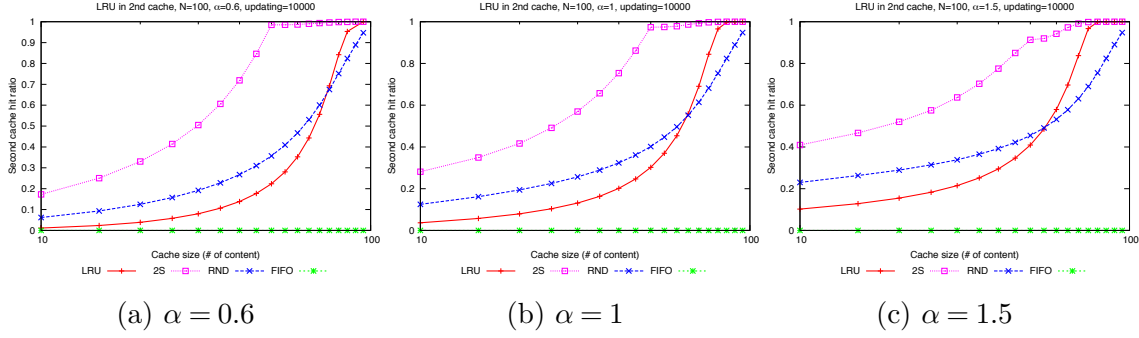


Figure A.7: Second cache (LRU) hit ratio with different α , $N=100$

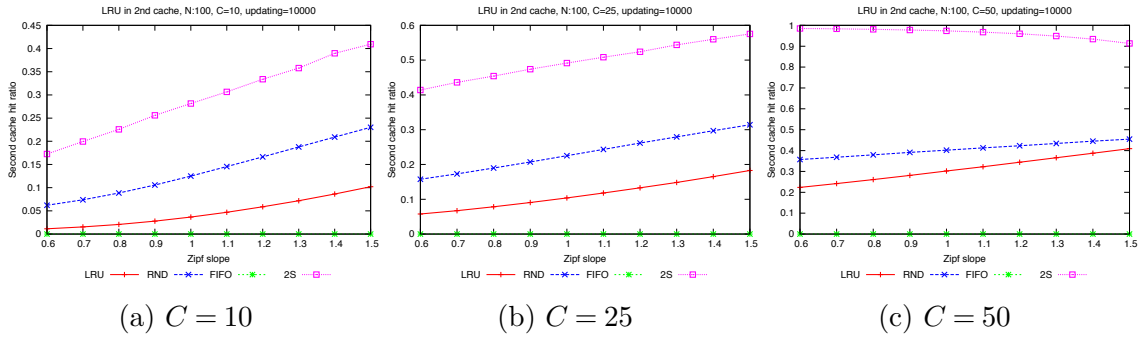


Figure A.8: Second cache (LRU) hit ratio with different cache sizes, $N=100$

RANDOM on Second Cache

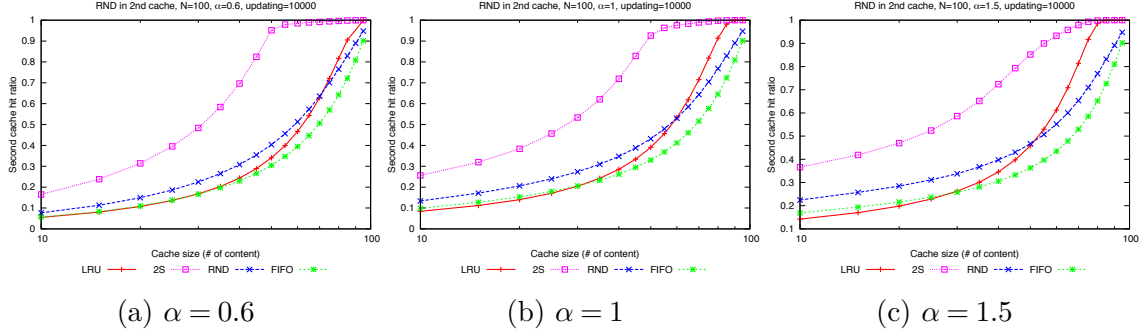


Figure A.9: Second cache (RND) hit ratio with different α , $N=100$

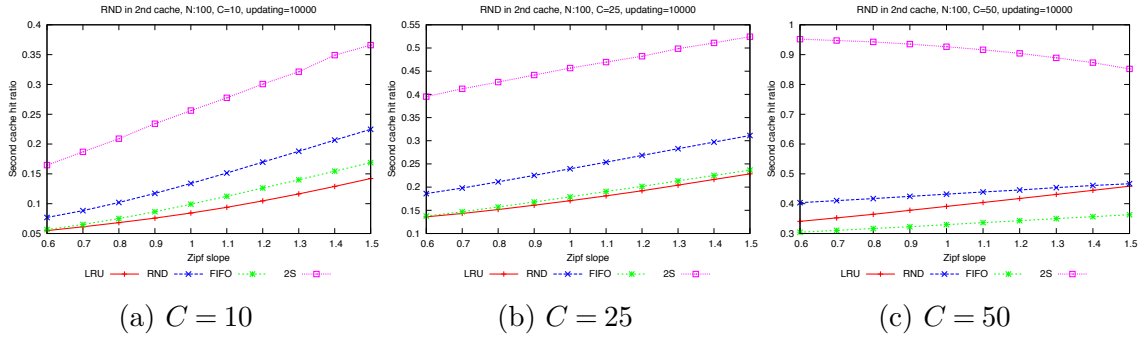


Figure A.10: Second cache (RND) hit ratio with different cache sizes, $N=100$

A.1.3 The First Cache Hit Ratio

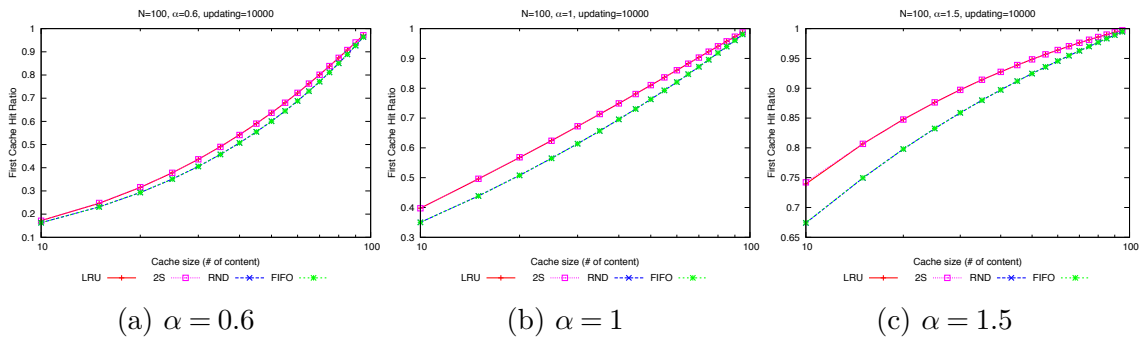


Figure A.11: First cache hit ratio with different α , $N = 100$

APPENDIX A. SYNTHETIC WORKLOAD FOR TWO-STATE POLICY

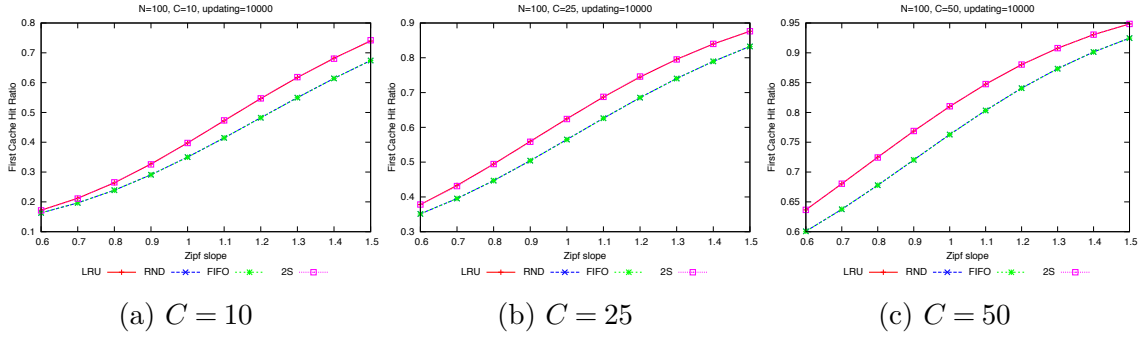


Figure A.12: First cache hit ratio with different cache sizes, $N=100$

A.2 Content Number of 1000

A.2.1 Stack Distance

Average

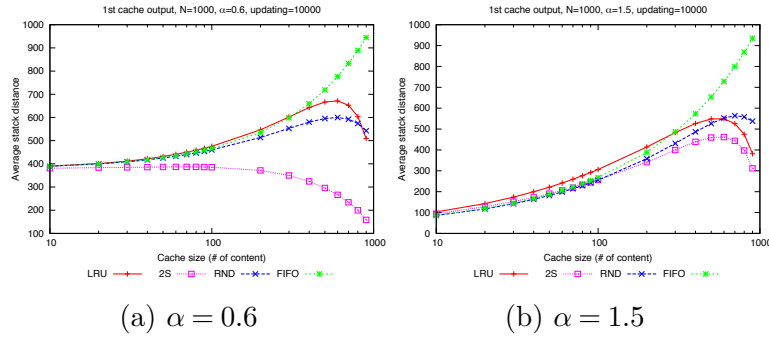


Figure A.13: Average stack distance with different α , $N=1000$

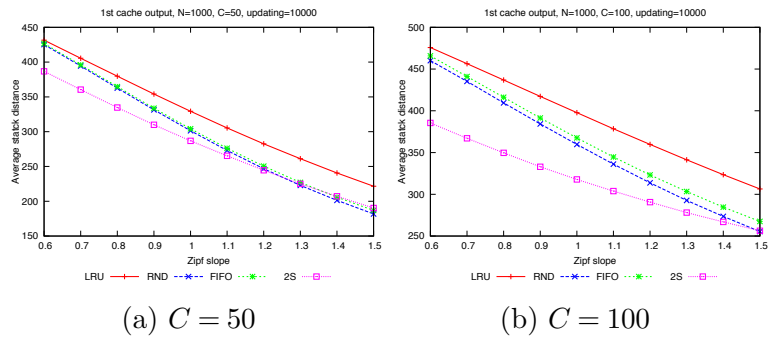


Figure A.14: Average stack distance with different cache sizes, $N=1000$

Minimum

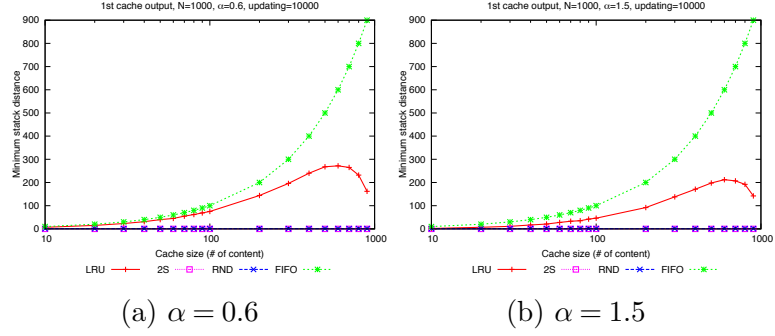


Figure A.15: Minimum stack distance with different α , $N=1000$

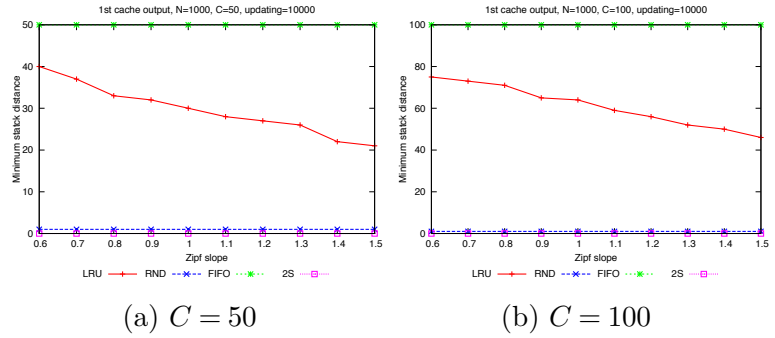


Figure A.16: Minimum stack distance with different cache sizes, $N=1000$

A.2.2 The Effect on Overall Cache Hit Ratio

FIFO on Second Cache

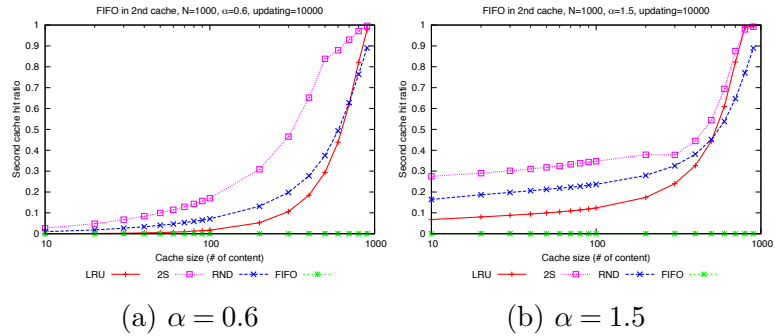


Figure A.17: Second cache (FIFO) hit ratio with different α , $N=1000$

APPENDIX A. SYNTHETIC WORKLOAD FOR TWO-STATE POLICY

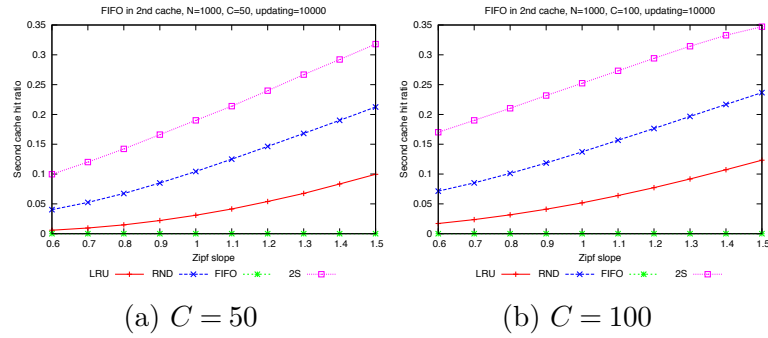


Figure A.18: Second cache (FIFO) hit ratio with different cache sizes, $N=1000$

LRU on Second Cache

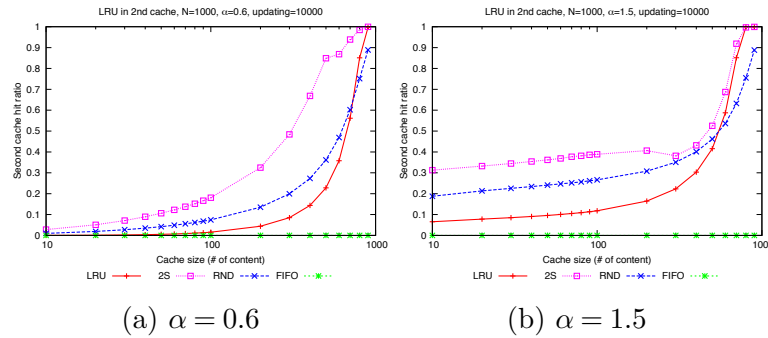


Figure A.19: Second cache (LRU) hit ratio with different α , $N=1000$

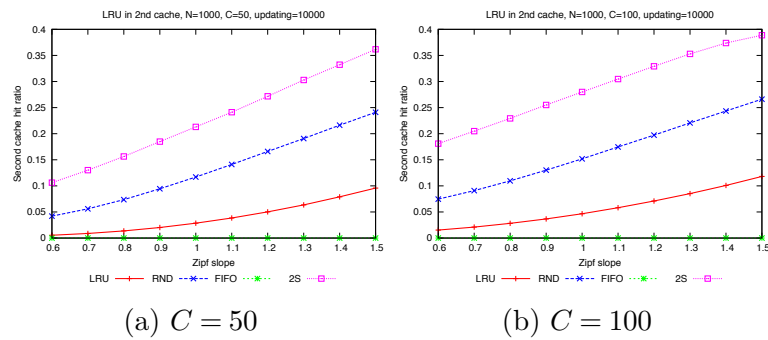


Figure A.20: Second cache (LRU) hit ratio with different cache sizes, $N=1000$

RANDOM on Second Cache

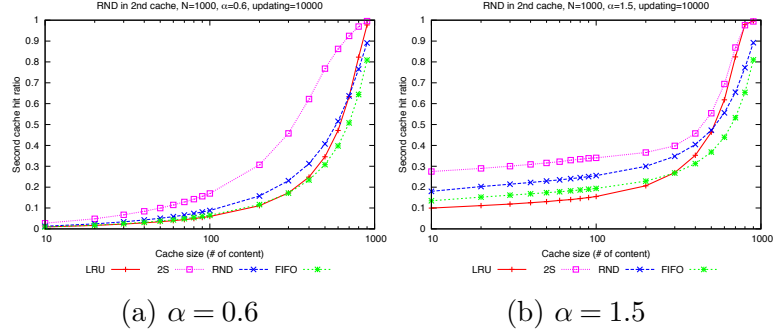


Figure A.21: Second cache (RND) hit ratio with different α , $N=1000$

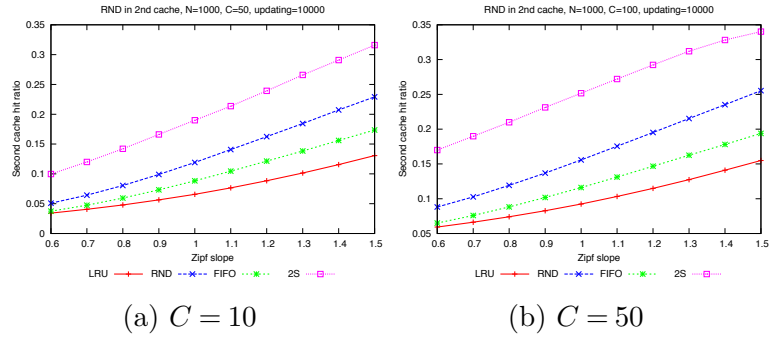


Figure A.22: Second cache (RND) hit ratio with different cache sizes, $N=1000$

A.2.3 The First Cache Hit Ratio

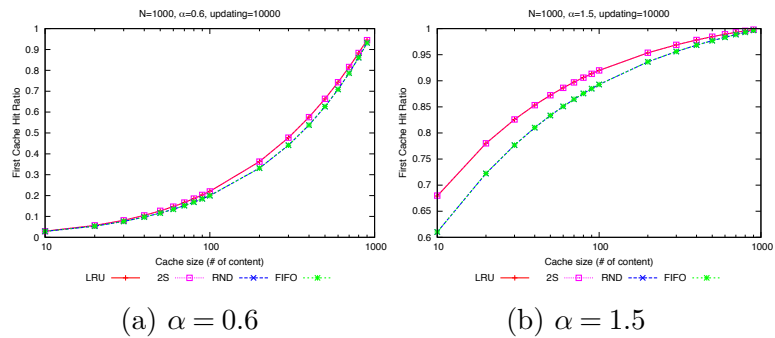


Figure A.23: First cache hit ratio with different α , $N = 1000$

APPENDIX A. SYNTHETIC WORKLOAD FOR TWO-STATE POLICY

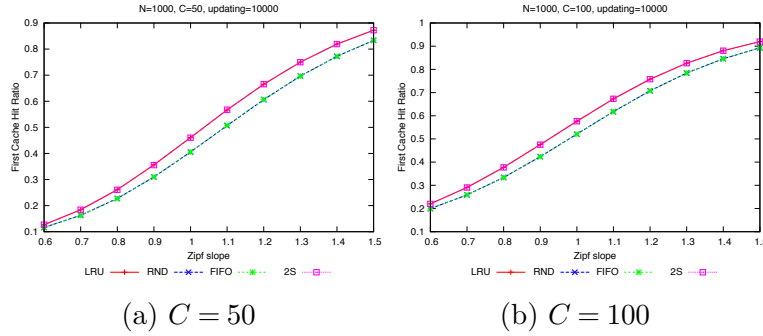


Figure A.24: First cache hit ratio with different cache sizes, N=1000

A.3 Content Number of 10000

A.3.1 Stack Distance

Average

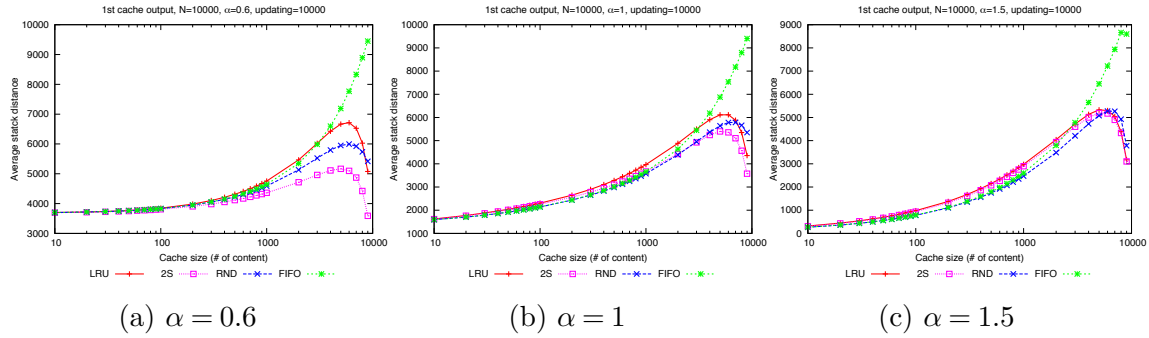


Figure A.25: Average stack distance with different α , N=100

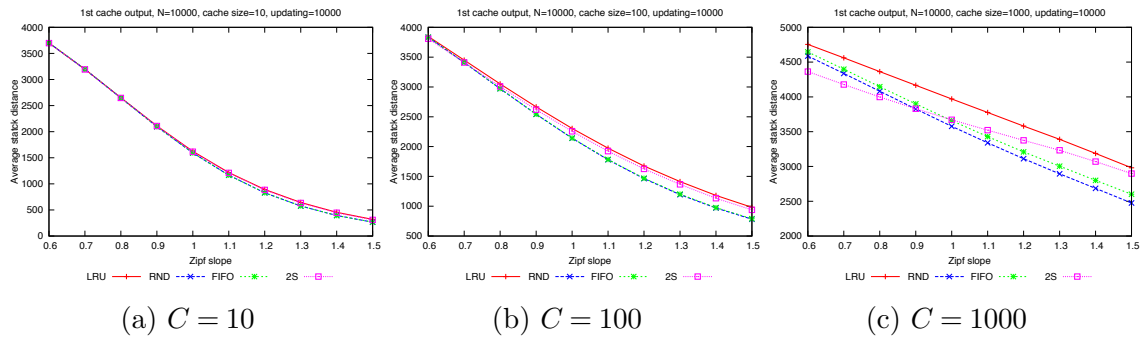


Figure A.26: Average stack distance with different cache sizes, N=1000

APPENDIX A. SYNTHETIC WORKLOAD FOR TWO-STATE POLICY

Minimum

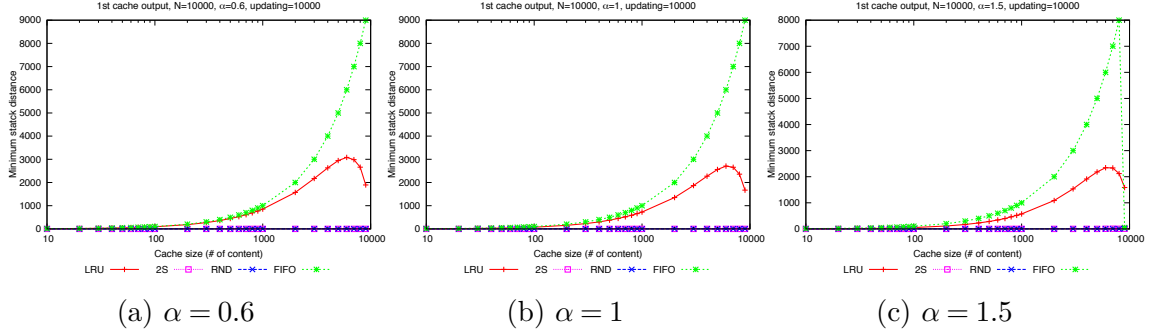


Figure A.27: Minimum stack distance with different α , $N=10000$

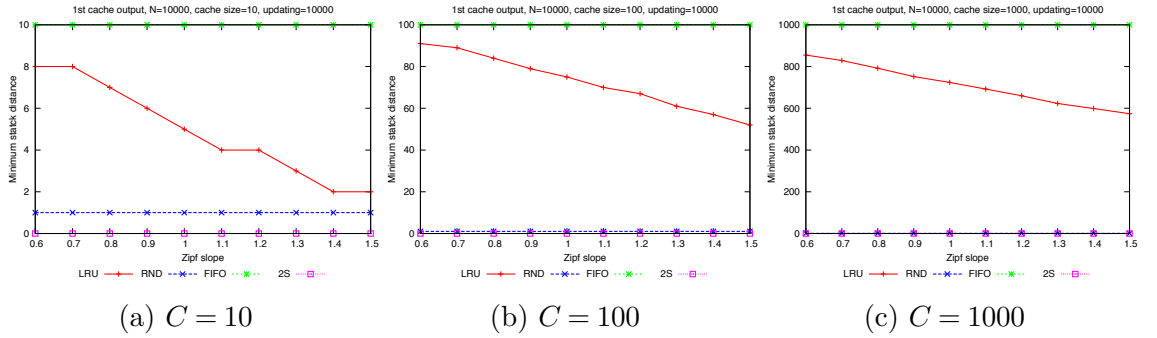


Figure A.28: Minimum stack distance with different cache sizes, $N=10000$

A.3.2 The Effect on Overall Cache Hit Ratio

FIFO on Second Cache

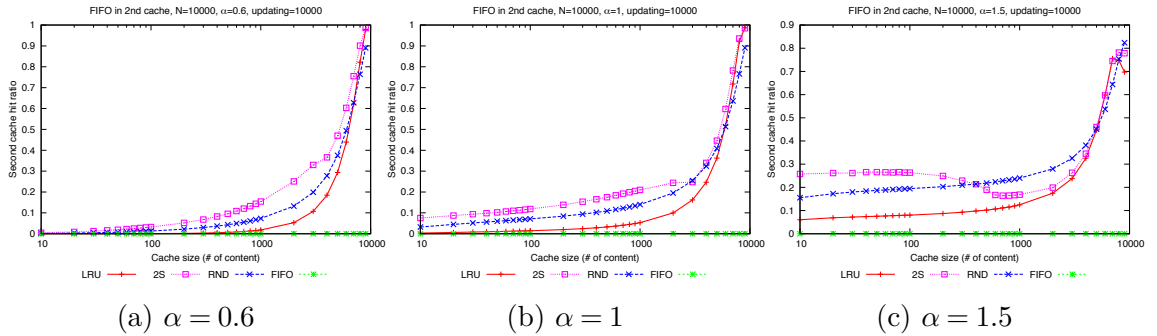


Figure A.29: Second cache (FIFO) hit ratio with different α , $N=10000$

APPENDIX A. SYNTHETIC WORKLOAD FOR TWO-STATE POLICY

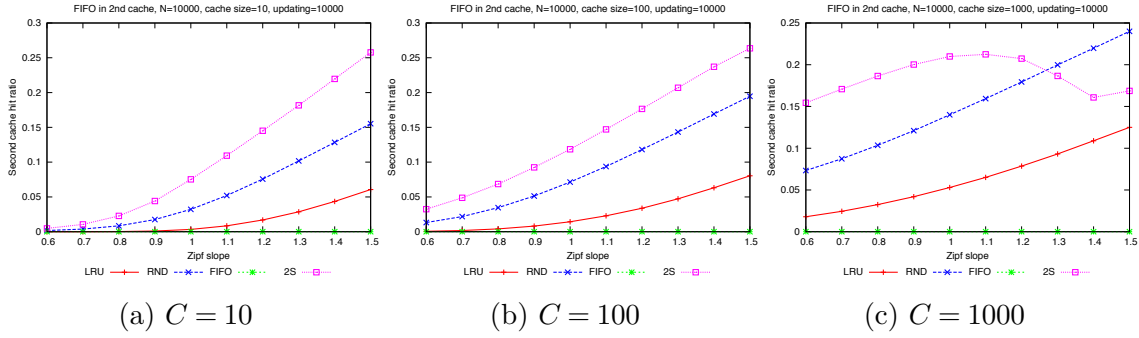


Figure A.30: Second cache (FIFO) hit ratio with different cache sizes, $N=10000$

LRU on Second Cache

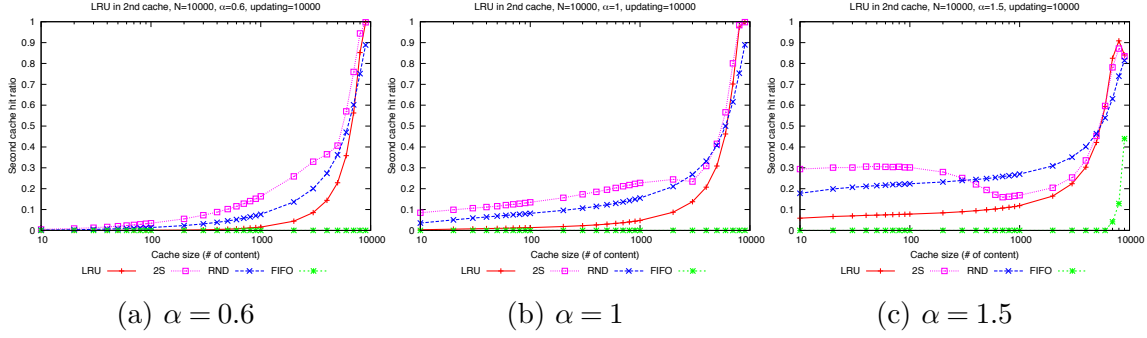


Figure A.31: Second cache (LRU) hit ratio with different α , $N=10000$

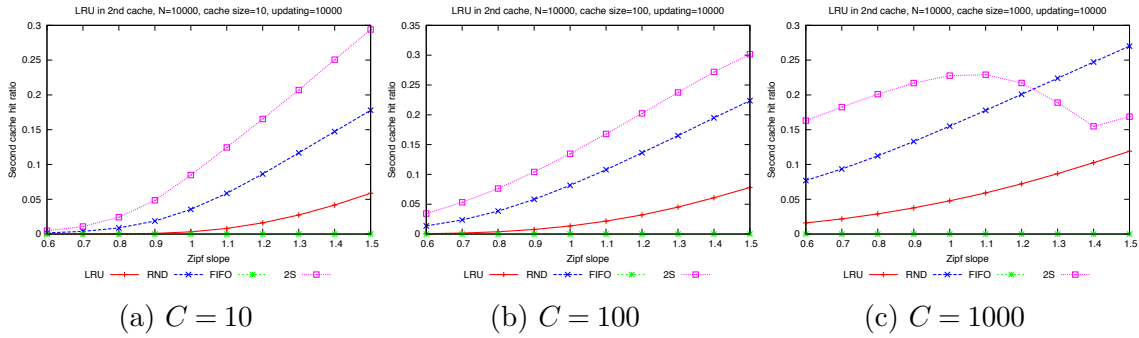


Figure A.32: Second cache (LRU) hit ratio with different cache sizes, $N=10000$

RANDOM on Second Cache

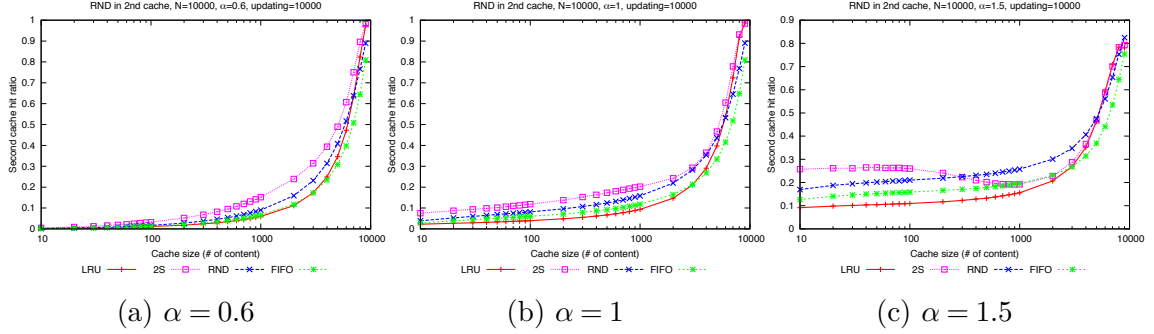


Figure A.33: Second cache (RND) hit ratio with different α , $N=10000$

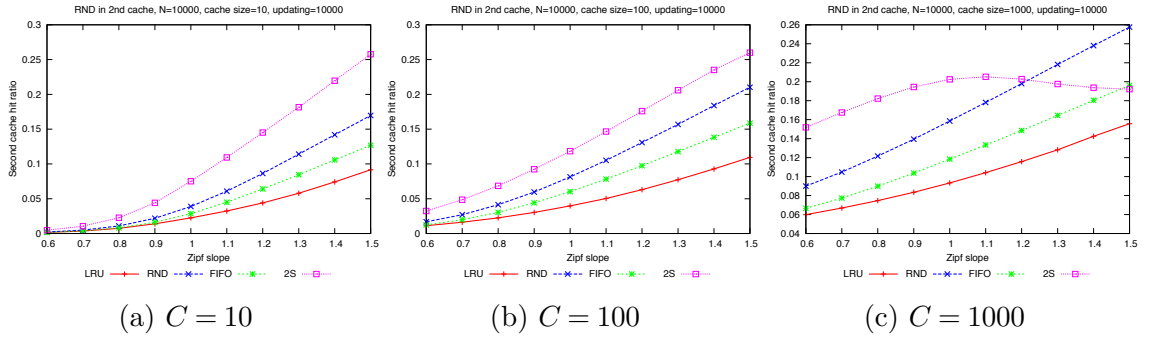


Figure A.34: Second cache (RND) hit ratio with different cache sizes, $N=10000$

A.3.3 The First Cache Hit Ratio

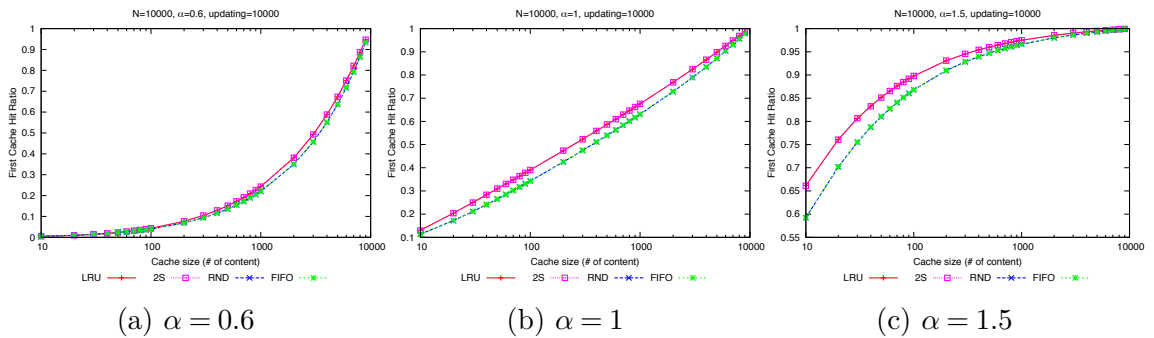


Figure A.35: First cache hit ratio with different α , $N = 10000$

APPENDIX A. SYNTHETIC WORKLOAD FOR TWO-STATE POLICY

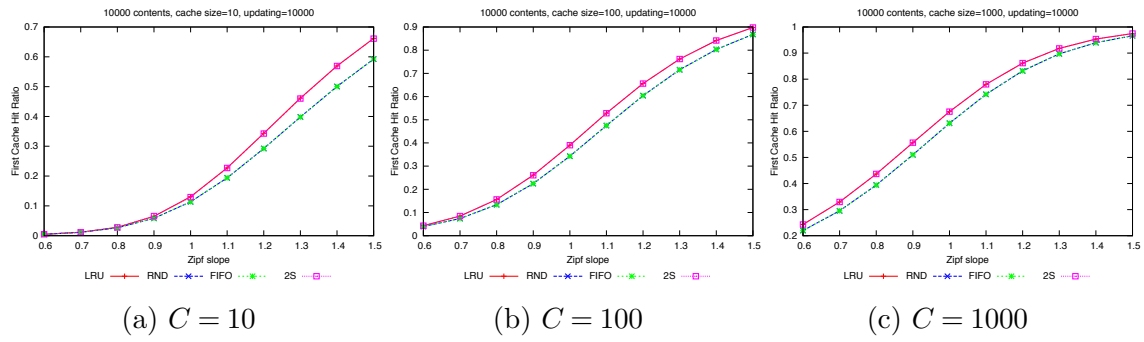


Figure A.36: First cache hit ratio with different cache sizes, $N=10000$

Appendix B

Trace Based Workload for Two-State Policy

In the previous appendix, we show that two properties of two-state policy are valid under IRM assumption. The objective of this appendix is to evaluate the same properties with trace-based simulation. In Section 3.2.3, we discussed that providing better opportunity for the core routers by using the two-state policy at edge routers is valid for some of the traces. In this appendix, we show that the property also valid for the rest of our traces but the second property, obtaining similar hit ratio as LRU, is not valid for trace-based evaluation. We discussed that this is due to the one-timer contents. We solved this issue with three-state policy that is evaluated in the next two appendixes.

B.1 Stack Distance

B.1.1 Average

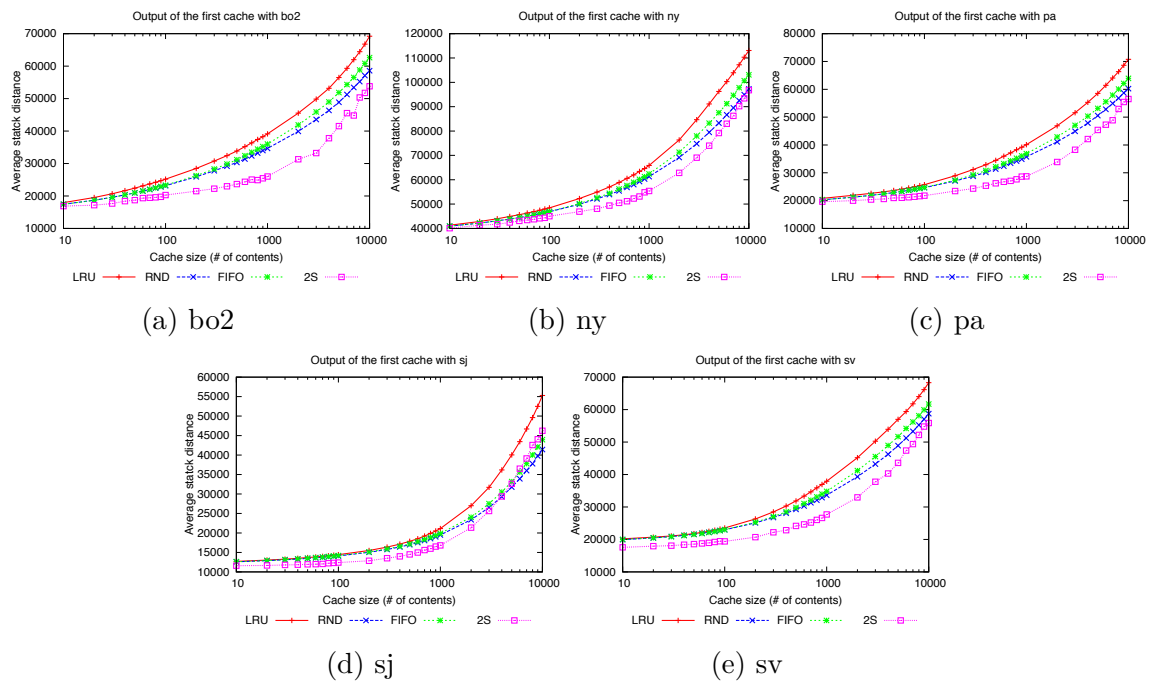


Figure B.1: Average stack distance with different traces

APPENDIX B. TRACE BASED WORKLOAD FOR TWO-STATE POLICY

B.1.2 Minimum

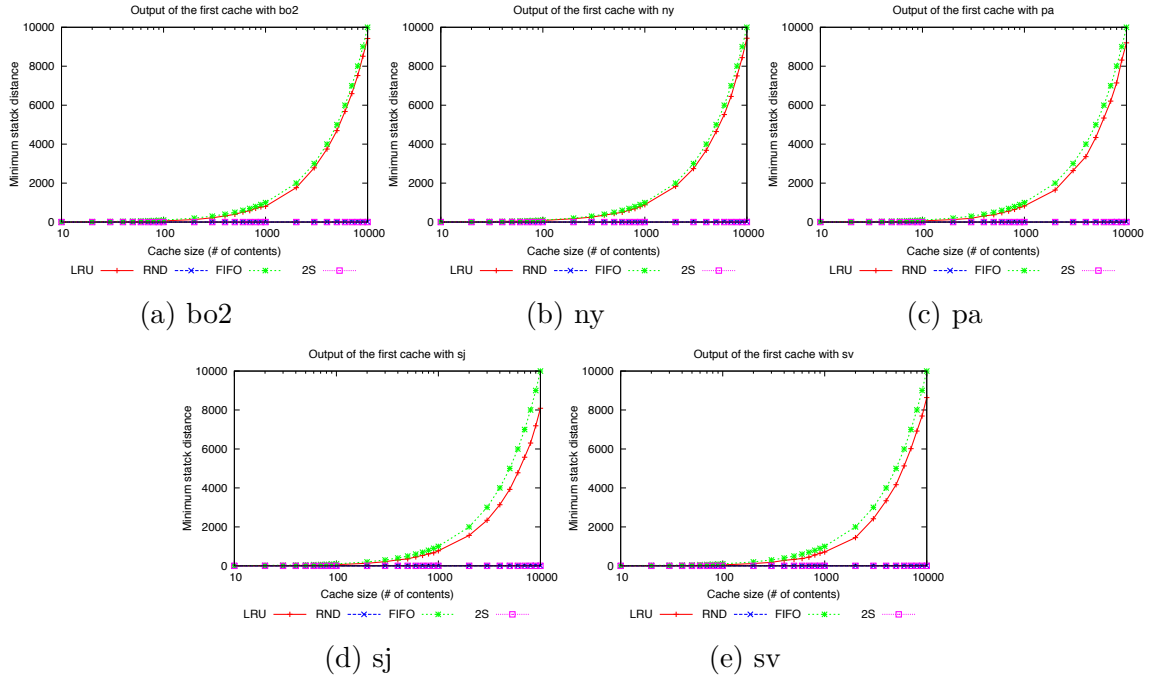


Figure B.2: Minimum stack distance with different traces

B.2 Benefit for Overall Hit Ratio

B.2.1 RANDOM on Second Cache

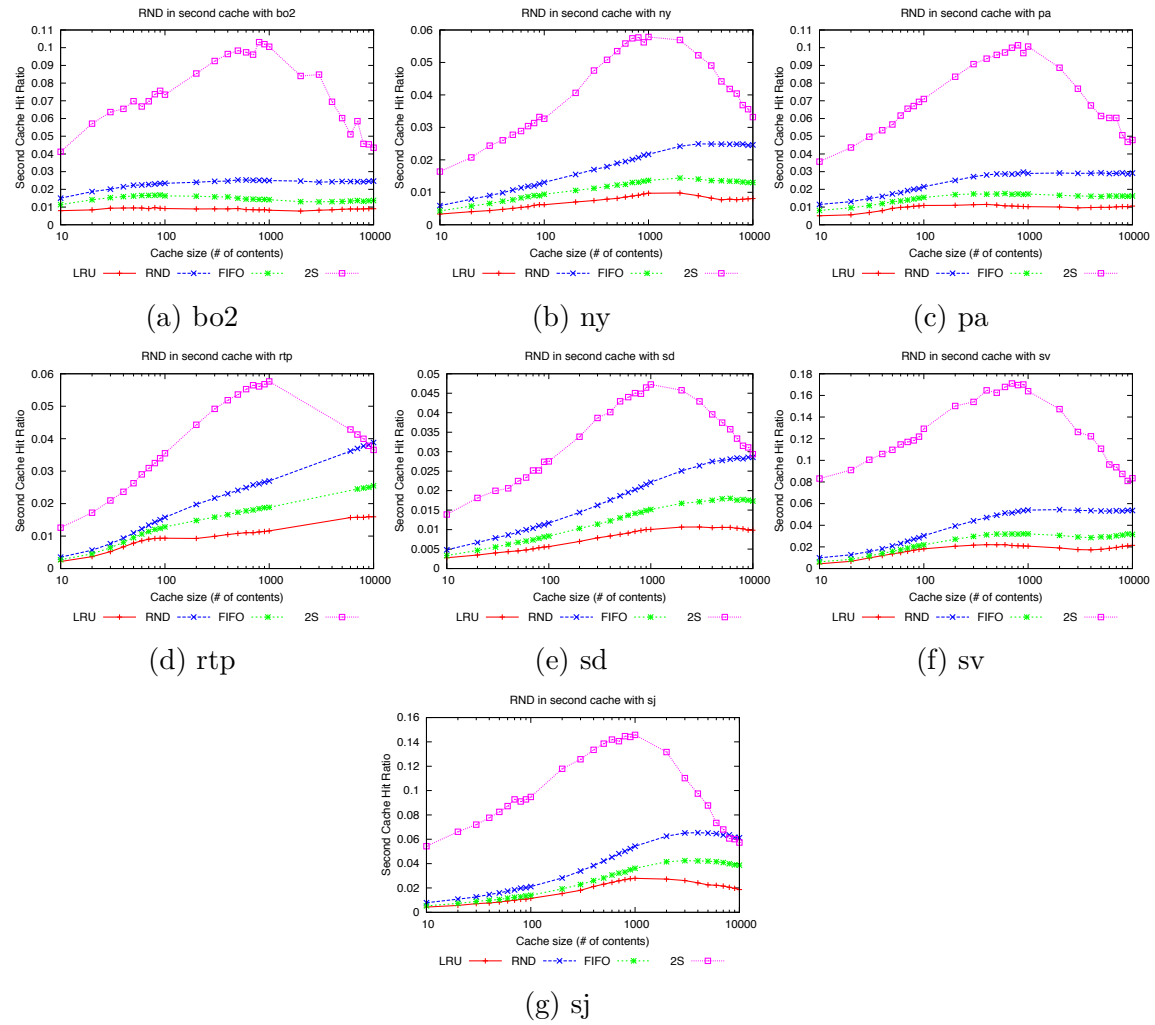


Figure B.3: Second cache (RND) hit ratio with different traces

B.2.2 LRU on Second Cache

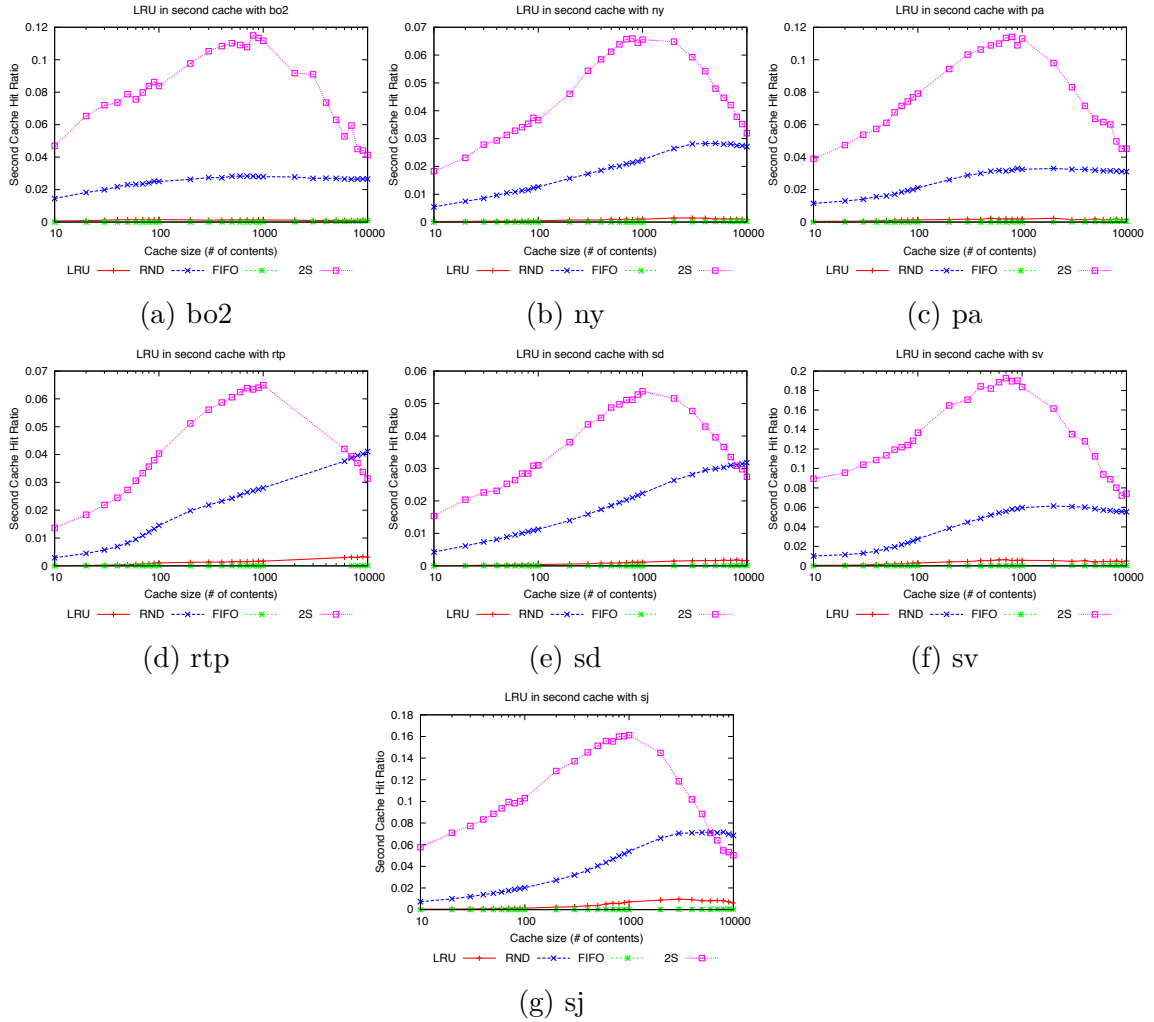


Figure B.4: Second cache (LRU) hit ratio with different traces

B.2.3 FIFO on Second Cache

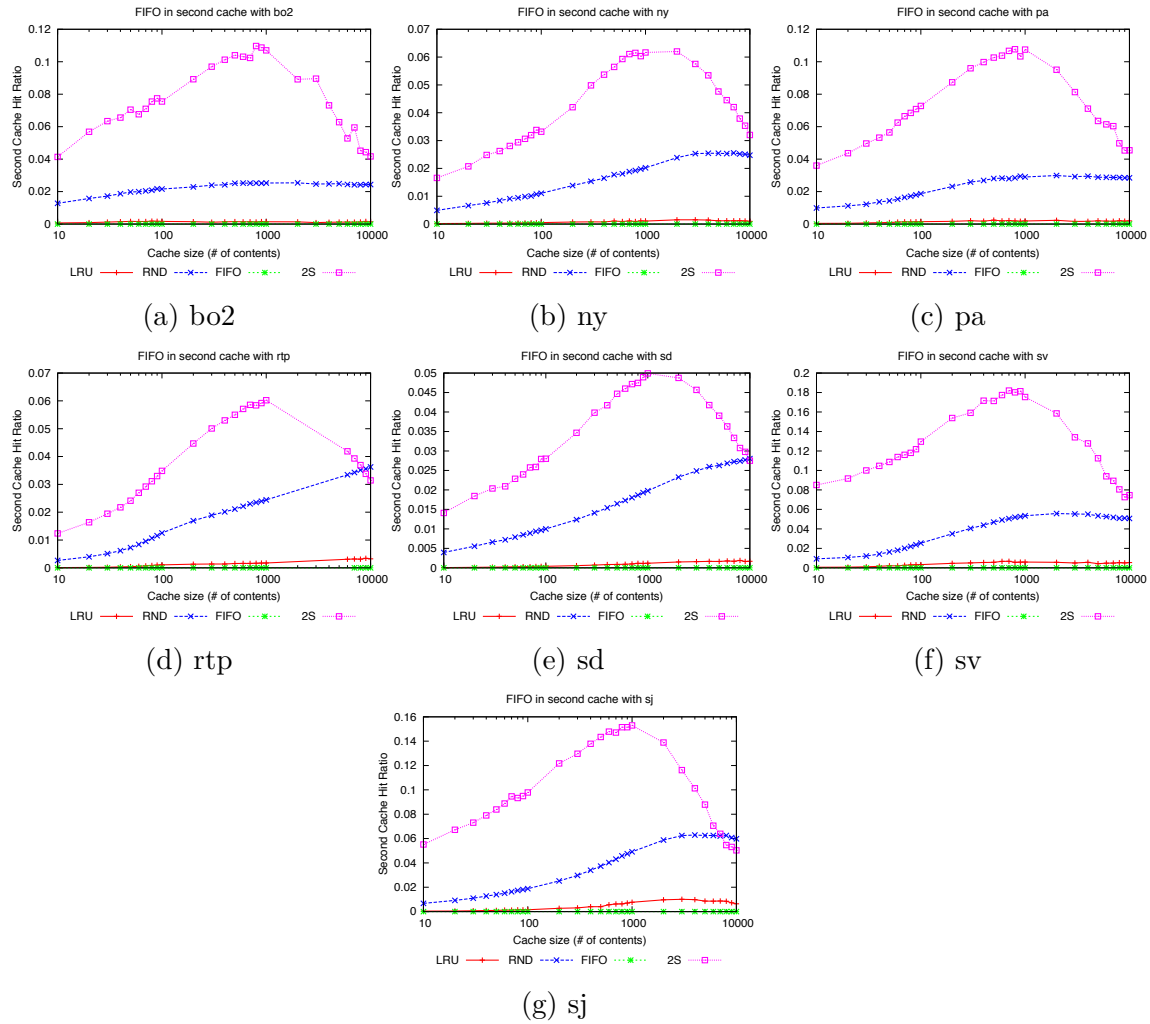


Figure B.5: Second cache (FIFO) hit ratio with different traces

B.3 The First Cache Hit Ratio

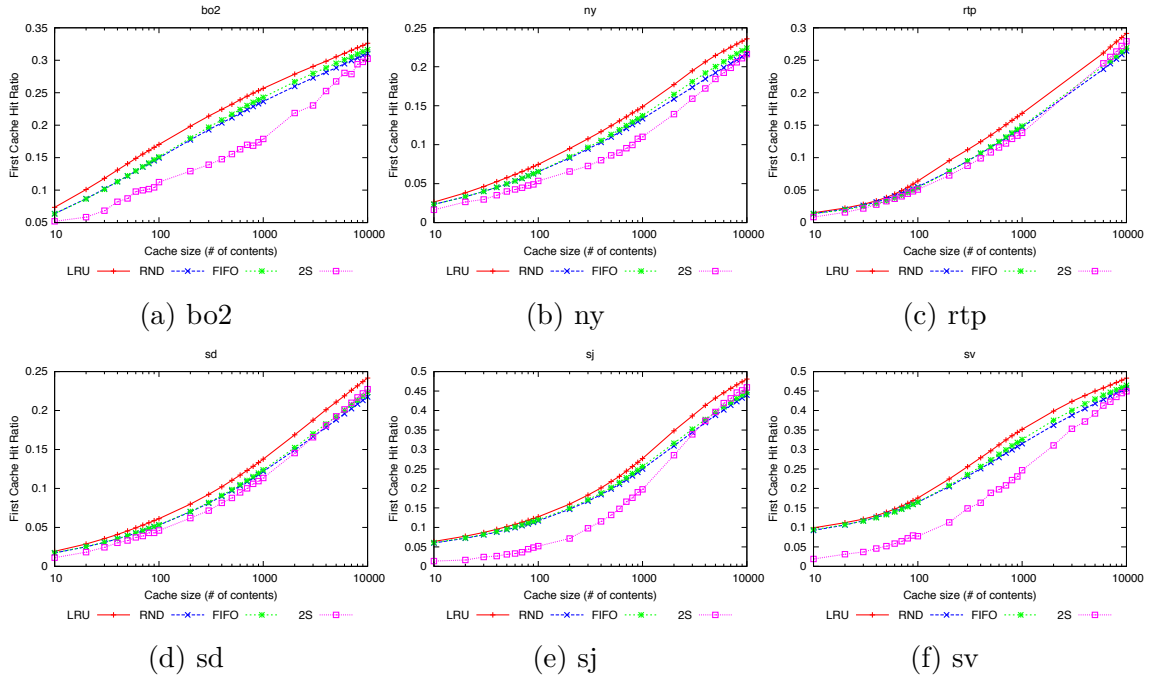


Figure B.6: First cache hit ratio with different traces

Appendix C

Synthetic Workload for Three-State Policy

The objective of this appendix is to show that three-state policy obtains higher hit ratio than two-state for a standalone cache under IRM assumption. We use IRM without one-timer content to show that three-state can capture popular contents almost as good as LFU. In the next appendix, we will show that three-state can obtain high hit ratio for the trace-based simulation as well. We use the same setting as described in Appendix A.

C.1 Content Number of 100

C.1.1 The First Cache Hit Ratio

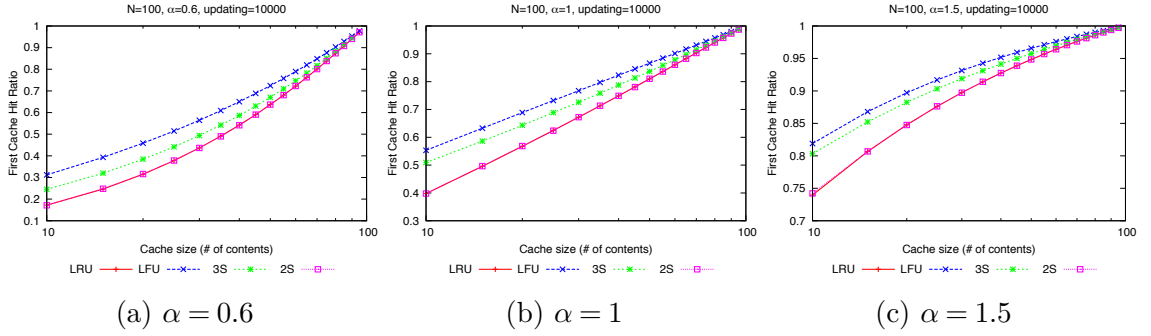


Figure C.1: First cache hit ratio with different α , $N = 100$

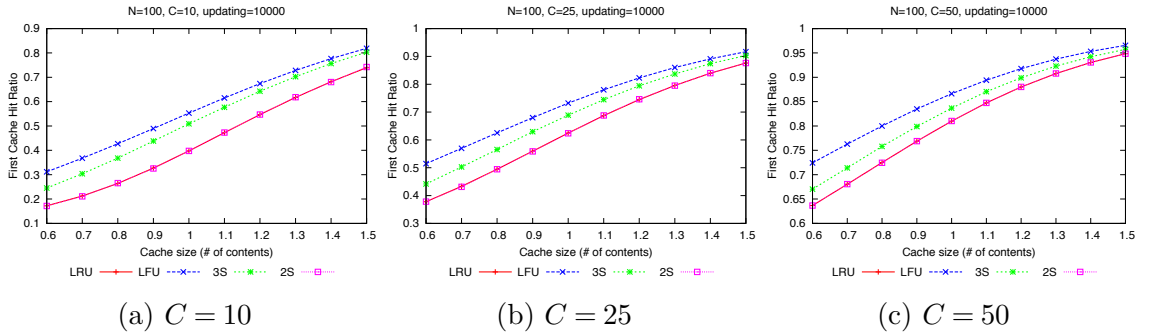


Figure C.2: First cache hit ratio with different cache sizes, $N=100$

C.2 Content Number of 1000

C.2.1 The First Cache Hit Ratio

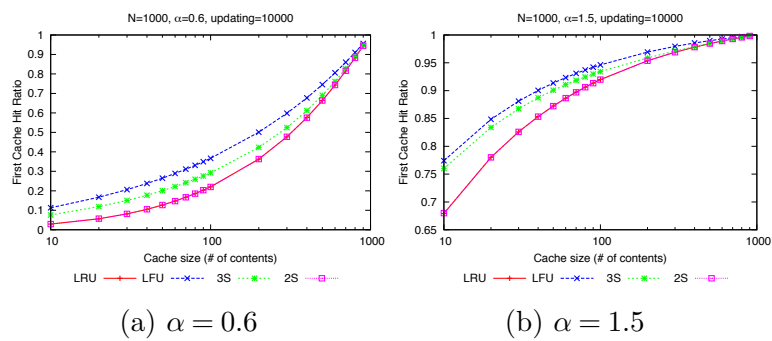


Figure C.3: First cache hit ratio with different α , $N = 1000$

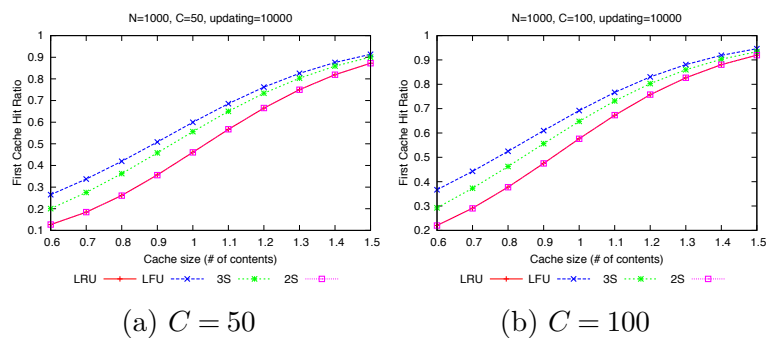


Figure C.4: First cache hit ratio with different cache sizes, $N=1000$

C.3 Content Number of 10000

C.3.1 The First Cache Hit Ratio

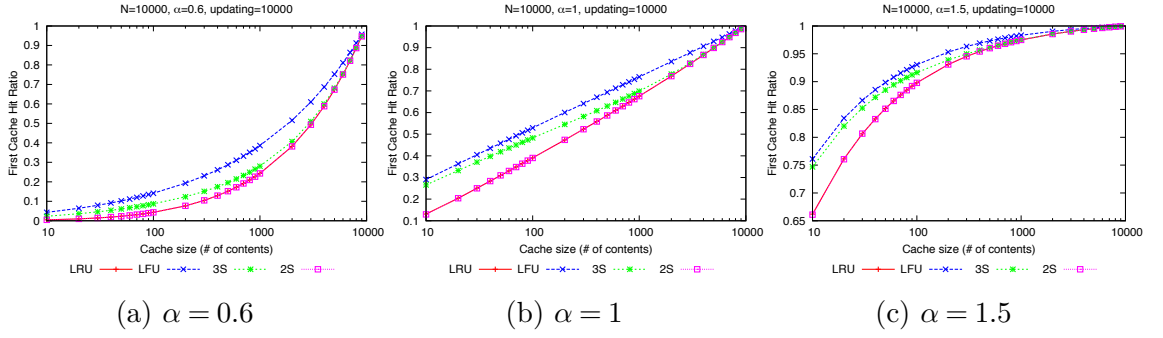


Figure C.5: First cache hit ratio with different α , $N = 10000$

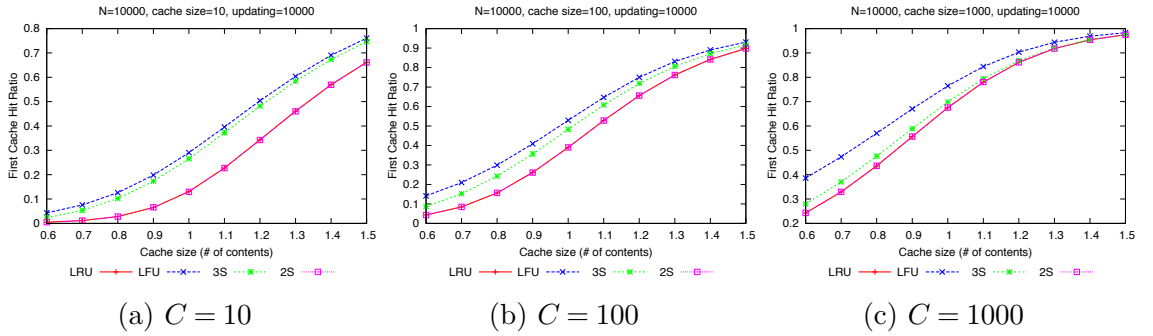


Figure C.6: First cache hit ratio with different cache sizes, $N=10000$

Appendix D

Trace Based Workload for Three-State Policy

In this appendix and through trace-based simulation, we show that the three-state overcomes the one-timer problem of two-state and obtains high hit ratio for a standalone cache.

D.1 The First Cache Hit Ratio

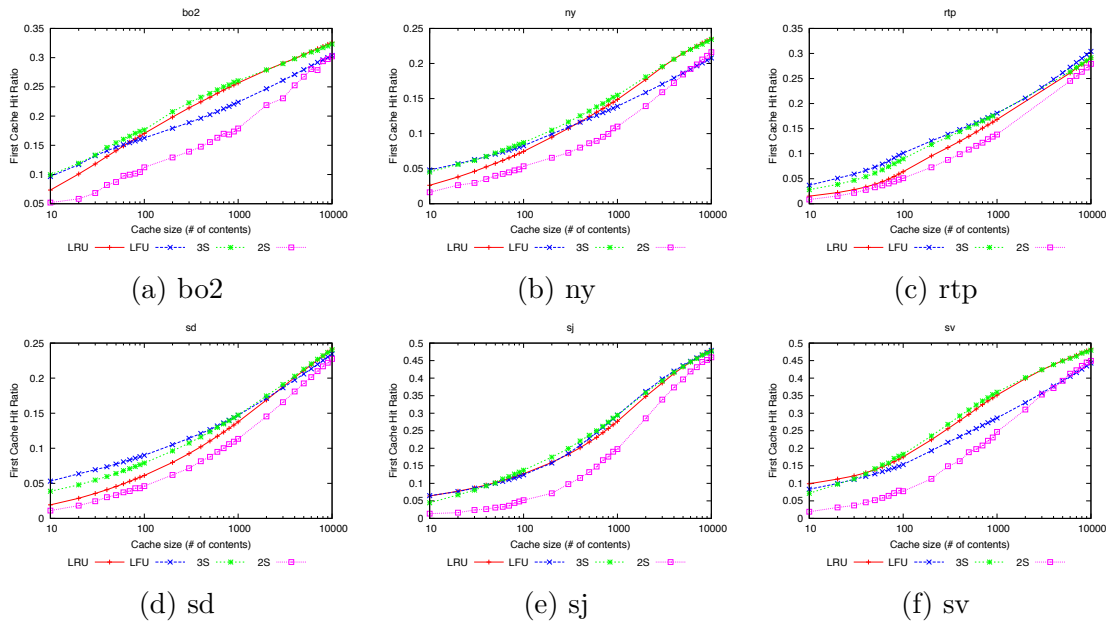


Figure D.1: First cache hit ratio with different traces