Department of Computer Science
Series of Publications A
Report A-2015-1

Content, Topology and Cooperation in In-network Caching

Liang Wang

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public examination in Auditorium CK112, Exactum building, Kumpula, Helsinki on March 27th, 2015 at 12 o'clock noon.

University of Helsinki
Finland

Supervisor
Jussi Kangasharju, University of Helsinki, Finland

Pre-examiners
Bin Liu, Tsinghua University, China
Guillaume Urvoy-Keller, Université Nice Sophia Antipolis, France

Opponent
Antonio Carzaniga, Università della Svizzera italiana, Switzerland

Custos
Jussi Kangasharju, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911, telefax: +358 9 876 4314

ISSN 1238-8645
ISBN 978-951-51-0824-1 (paperback)
ISBN 978-951-51-0825-8 (PDF)
Computing Reviews (1998) Classification: C.2, C.2.1, C.2.2
Helsinki 2015
Unigrafia

Content, Topology and Cooperation in In-network Caching

Liang Wang

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
liang.wang@helsinki.fi
http://cs.helsinki.fi/liang.wang

PhD Thesis, Series of Publications A, Report A-2015-1
Helsinki, February 2015, 190 pages
ISSN 1238-8645
ISBN 978-951-51-0824-1 (paperback)
ISBN 978-951-51-0825-8 (PDF)

Abstract

In-network caching aims at improving content delivery and alleviating pressures on network bandwidth by leveraging universally networked caches. This thesis studies the design of cooperative in-network caching strategy from three perspectives: content, topology and cooperation, specifically focuses on the mechanisms of content delivery and cooperation policy and their impacts on the performance of cache networks.

The main contributions of this thesis are twofold. From measurement perspective, we show that the conventional metric hit rate is not sufficient in evaluating a caching strategy on non-trivial topologies, therefore we introduce footprint reduction and coupling factor, which contain richer information. We show cooperation policy is the key in balancing various tradeoffs in caching strategy design, and further investigate the performance impact from content per se via different chunking schemes.

From design perspective, we first show different caching heuristics and smart routing schemes can significantly improve the caching performance and facilitate content delivery. We then incorporate well-defined fairness metric into design and derive the unique optimal caching solution on the Pareto boundary with bargaining game framework. In addition, our study on the functional relationship between cooperation overhead and neighborhood size indicates collaboration should be constrained in a small neighborhood due to its cost growing exponentially on general network topologies.

Computing Reviews (1998) Categories and Subject Descriptors:
C.2 Computer-Communication Networks
C.2.1 Network Architecture and Design
C.2.2 Network Protocols

General Terms:
Algorithm, Theory, Design

Additional Key Words and Phrases:
Information-centric networking, cache networks, collaborative caching, complexity analysis, graph theory, game theory, optimization theory

Acknowledgements

My foremost gratitude goes to my supervisor, Prof. Jussi Kangasharju, who has guided me through my PhD research with his exceptional patience and wisdom. I also thank Prof. Bin Liu, Prof. Guillaume Urvoy-Keller and Prof. Antonio Carzaniga for their great reviews on the thesis.

My gratitude extends to my coauthors, colleagues and friends, for their warm and unflagging support during my nearly six-year stay in Finland. The financial support from Future Internet Graduate School, Department of Computer Science effectively shielded me from mundane matters.

I am much obliged to both my Chinese and Finnish families for their unconditional trust and everlasting love. I thank my cat Sissi for her great help in writing the thesis by sleeping on my keyboard.

I thank my beloved parents Yulin Wang and Fengjin Liang for giving me wings to fly so that I can explore the unknown; I thank my beloved wife Maria Wang for bringing color to my life so that I can paint the future.

Liang Wang

Helsinki, Finland
January, 2015

Contents

Chapter 1

Introduction

1.1 Background and Motivation

The emergence of Information-Centric Networking (ICN) is mostly attributed to the fact that both academia and industry realized that nowadays our use of the Internet is mainly for content distribution. The original point-to-point paradigm of the Internet, which can be traced back to the very beginning of computer networks in 1960s and '70s, is inept in the new context wherein rich multimedia content prevails and dominates. As a promising candidate for future Internet, ICN proposes a clean-slate redesign which builds the network infrastructure around the content and shifts the current Internet from sender-driven point-to-point communication to receiver-driven content distribution paradigm. This new alternative to the future network architecture aims at ameliorating many issues confronting the current Internet such as content distribution efficiency, congestion, security, and etc. While there are several independent proposals [1–3] in the ICN literature, the central idea contained is essentially the same – *accessing content by name*; and *universal caching*.

However, ICN is not a silver bullet, and many research and engineering challenges are still awaiting effective solutions. E.g. the number of content objects is orders of magnitude larger than the number of routers in the Internet [4]. Conservative estimate based on the current size of web content (i.e. number of URLs Google has indexed) indicates any ICN system is expected to manage about 10^{12} objects. The huge disparity between enormous space of content names and scarce routers' resources still remains as one of the biggest challenges for the practical large-scale deployment of ICN. Among the new challenges posed by ICN on routing, content naming, content addressing and etc., universal caching, which is often referred as

in-network caching, has aroused significant research interest these years. One reason is caching has long been recognized as an effective technique in solving scalability issues for large-scale systems. Another reason is caching directly influences the efficiency of content distribution which further impacts end-user experience in many Internet applications, rendering it an indispensable component in ICN architecture.

The principal task of in-network caching research is analyzing and designing effective caching strategies to improve the system performance, which is also the focus of this thesis. Unfortunately, the prior experience we learned from the long history of edge cache deployment only provides limited help in the new context. Shifting from a single stand-alone cache to a cache network invalidates a large part of our prior knowledge, and drastically changes the way we measure and design a caching system as it used to be, which urges us to understand the fundamental difference between the traditional solution and its novel substitute. Besides that distributed caching strategy design per se is already a challenging job, the proper measurement metrics and sound methodology we shall adopt in evaluating an ICN system are not trivial at all in the first place. In short, we want to answer the following questions in this thesis:

1. What metrics should we use in evaluating an in-network caching strategy? How should we describe the relationship among content, topology and cooperation policy with the given metrics?

2. How should we incorporate multiple metrics into the design goals and how to balance the tradeoff in a caching strategy via cooperation?

By simulation, heuristic design, mathematical modeling and analysis, this thesis puts its initial effort into understanding the key features that distinguish in-network caching from conventional edge caching, then further focuses on the study of cooperation policy in a caching strategy. The thesis approaches its goal from both empirical and theoretical perspectives.

We hope this thesis can make its original contribution to the ICN community, shed light on understanding the role of cooperation policy in distributed caching systems, and also provides some fundamental guidance in designing caching strategies for future Internet infrastructures.

1.2 Thesis Contributions

In a very general sense, this thesis devotes itself to understanding the relationship among the content, topology and cooperation policy in the caching

strategy design. Its major contributions are twofold as described in the following. The detailed contributions will be stated separately in the beginning of each individual chapter.

Firstly, from system measurement perspective, we show the conventional metric (byte) hit rate is not sufficient in terms of evaluating an in-network caching strategy. Other more complicated metrics such as footprint reduction, coupling factor and etc., which contain much richer information, should also be taken into account in order to more thoroughly describe a system. With a concise yet expressive optimization model (in Chapter 7), we illustrated the interplay of hit rate and footprint reduction under cooperation policy, showing how they become conflicting with each other on a nontrivial topology when system performance reaches its Pareto frontier. The results not only contribute a new metric – coupling factor, a scalar value containing the information from both content popularity and network topological properties, but also lead to the key insight that it is cooperation policy which couples the content with topology, and balances the tradeoff between intra- and inter-domain traffic. In other words, adopting multiple metrics implies the potentially infinite optimal caching strategies with varying tradeoffs (between hit rate and footprint reduction) on the Pareto boundary. From an ISP's perspective, hit rate and footprint reduction represent the savings on inter- and intra-domain traffic respectively. The results further imply that given the system resources are fully utilized, further reducing the inter-domain traffic will inevitably increase the intra-domain traffic.

Secondly, from protocol design perspective, based on the aforementioned important results, we focus on the analysis of cooperation policy from both experimental and analytical perspectives. The en-route heuristics with naive cooperation we designed in the beginning of this thesis is already able to significantly boost the performance before reaching the Pareto frontier. Combined with smart routing schemes, we show the efficiency of content delivery can be further improved. The more theoretical work in the later chapters abstracts the cooperation policy with a node's search radius and formally models the in-network caching as a Nash bargaining game. By translating the bargaining game to its equivalent convex optimization problem, we derive a solution in finding the unique optimal caching strategy which fulfills the requirements on Pareto optimality and well-defined fairness such as proportional fairness and max-min fairness. We also thoroughly analyze the communication cost of the derived distributed caching algorithm with various settings on neighborhood and types of graphs. More importantly and interestingly, as the content locality is observed in the re-

quest stream, our extensive simulations show that the similar property can also be found in nodes' cooperation, namely most of the interactions among the nodes occur in a small neighborhood, which is also referred as topological locality. Eventually, we reach the final conclusion that while the admission and eviction policy in a conventional caching algorithm aims at capturing the content locality to improve the hit rate, cooperation policy should be constrained in a small neighborhood and take advantage of the topological locality to reduce the traffic footprint.

1.3 Thesis Organization

This thesis contains an introductory chapter, followed by an overview of the recent work on ICN with a focus on in-network caching in Chapter 2. Chapter 3 discusses the methodology and metrics we developed and used throughout the thesis in the performance evaluation, also gives a detailed description on our experiment platform.

In Chapter 4, we present several en-route caching heuristics and neighborhood search technique, and also compare these naive cooperation policies to redundancy-elimination techniques. Chapter 5 compares several popular mobility schemes, and shows combining compact routing in ICN improves the content delivery efficiency and also solves its related mobility issue. Chapter 6 further develops the idea by enhancing the compact routing with prefix embedding and topology-aware hashing.

From Chapter 7 and afterwards, we put more focus on the analytical perspective. Chapter 7 contains the core idea of this thesis and depicts a birdview of the topic. With an optimization model, we first illustrate the effects of cooperation policy and topology on caching performance by correlating the content popularity with topological properties, namely where the popular content is deployed by cooperation policy. Then we further propose the coupling factor and the categorization of cooperation policies. Chapter 8 concentrates on the content itself, and builds an analytical model to show the optimal chunking is too expensive and only brings marginal benefits in practice. The results indicate the naive way currently adopted in dividing the content is sufficient to achieve nearly-optimal performance.

Based on the results obtained in the previous chapters, we model the in-network caching as a Nash bargaining game and solve its equivalent convex optimization problem in Chapter 9. The derived distributed caching algorithm is thoroughly analyzed and evaluated in the same chapter. Finally, Chapter 10 concludes the thesis and outlooks the future directions.

1.4 Published Work

The following research papers constitute the Chapter 3, 4, 5, 6, 7, 8 and 9 in this thesis. Research paper I, II, III, IV and V have been published, and VI is currently in submission.

I. L. Wang, S. Bayhan, and J. Kangasharju, "Effects of Cooperation Policy and Network Topology on Performance of In-network Caching," IEEE Communication Letters. IEEE, Vol.18, No.4, April 2014.

II. W. Wong, L. Wang, and J. Kangasharju, "Neighborhood Search and Admission Control in Cooperative Caching Networks," in the Proceedings of IEEE Globecom. IEEE, December 3-7 2012.

III. L. Wang, O. Waltari, and J. Kangasharju, "MobiCCN: Mobility Support with Greedy Routing in Content-Centric Networks," in the Proceedings of IEEE Globecom. IEEE, December 9-13 2013.

IV. S. Roos, L. Wang, T. Strufe, J. Kangasharju, "Enhancing Compact Routing in CCN with Prefix Embedding and Topology-Aware Hashing," in the Proceedings of ACM MobiCom workshop on MobiArch. ACM, September 7-11 2014.

V. L. Wang, S. Bayhan, and J. Kangasharju, "Optimal Chunking and Partial Caching in Information-Centric Networks," in Journal of Computer Communications. Elsevier, February 2015

VI. L. Wang and J. Kangasharju, "A Fair Collaborative Game on Cache Networks," in submission.

Chapter 2

Overview of ICN

In this chapter, we will give an overview of the cutting-edge research in ICN, with specific interest on the recent work in in-network caching measurement and design. We start the discussion with the brief history of ICN, list some important designs, then focus on its core architectural components.

Launched in 2000, TRIAD project [5, 6] is arguably the first pioneer work in ICN and is considered as a precursor of all the related work discussed in this thesis. In 2002, a similar point was expressed in [7], outlooked the future Internet should shift from point-to-point communication to delivery of named objects. Both designs focus on the basic content delivery by exploiting the existing DNS system, other critical aspects like security, caching and etc. were not covered. Unfortunately, these two prominent work did not capture enough attention in the research community since for a long period of time, there was very few follow-up work on the topic.

In 2007, another seminal work [3] which is known as Data-Oriented Networking Architecture (DONA), was published and gave the first comprehensive description on a clean-slate redesign of the Internet. In a couple of years' time, DONA ignited several important follow-up work [1, 8–15] with the important ones listed as follows:

1. Content-Centric Networking (CCN) [1] in the Named Data Networking (NDN) project, published in 2009.

2. Publish-Subscribe Internet Routing Paradigm (PSIRP) [8] in Publish-Subscribe Internet Technology (PURSUIT) project, published in 2008.

3. Network of Information (NetInf) [9] in Scalable and Adaptive Internet Solutions (SAIL) project (previously known as 4WARD [16]), published in 2008.

These important work eventually brought the ICN into the mainstream networking research. All these designs have fairly complex architecture and are quite different even in their terminologies, but their common goal is the same – providing efficient, reliable and safe content delivery with an open and general purpose framework by constructing network infrastructure around the content. With this goal, these commonly called ICN approaches resemble with each other in several fundamental principles we will discuss one by one in the following. As supplements, [4, 17–21] give very thorough overviews on different ICN designs by delving into their mechanical details.

2.1 Operation Primitives

We have to mention publish/subscribe (pub/sub) paradigm before introducing ICN operation primitives. Because in some sense, ICN is just an application of pub/subs paradigm at Internet-scale. The concept of pub/sub was already proposed over two decades ago (in 1980s) [22] and had a lot of successful applications in different services and systems. The pub/sub paradigm decouples the responses and requests in both time and space by applying two operation primitives: `PUBLISH` and `SUBSCRIBE`. As a result of decoupling, the content delivery from the content producer to content consumer can be done in both asynchronous and location-independent manner.

Pub/sub paradigm has a profound influence on future network architectures, including ICN. This explains the resemblance of the basic primitives in all ICN proposals to those in pub/sub system. E.g. CCN uses `REGISTER` and `INTEREST`, and Curling uses `PUBLISH` and `CONSUME`, PSIRP further directly borrows the pub/sub terminology without any change. Therefore, ICN inherits the flexibility of pub/sub system in content delivery by adopting its operation primitives. In addition, ICN provides extra semantic to the pub/sub primitives by allowing a content consumer to explicitly request an object which has been published before.

There are fewer research activities in network primitives comparing to other aspects in ICN. To solve certain problems, most work exploits and adds more semantics to the existing primitives (often by adding new parameters) rather than introducing new ones. Hence, the operation primitives are very stable and likely to remain the same even in the foreseeable future.

2.2 Naming & Security

Content naming is a core component in ICN architecture, and it has narrower request semantics comparing to pub/sub system where content can be tagged with various keywords. Because individual content is identified and accessed by its unique name and independent of its location and container, the corresponding security mechanisms must be provided to guarantee the data *integrity*, *authenticity* and *provenance*. Such mechanism is usually implemented via verifiable binding between content and name, e.g. signature with public key or other cryptography techniques. The ICN security model assumes that the requester knows both content name and its publisher's public key, therefore is responsible for checking the authenticity, provenance and integrity of the content.

As we can see from above, an ideal naming scheme is expected to be unique, secure, scalable, user friendly and location-independent. In reality, it is difficult if not impossible to find such a scheme satisfying all these requirements. While all the ICN designs basically adopt the same network primitives, they diverge in the choice of naming schemes. There are three naming schemes exist – *flat naming*, *hierarchical naming* and *attribute-based naming* [14], with the previous two (*flat* and *hierarchical*) dominate the ICN literature, and are advocated by [23] and [24] respectively. The diametrically opposing viewpoints expressed in [23] and [24] originates from the different concerns and tradeoff on security, efficiency and scalability.

Similar to the current DNS, hierarchical naming often uses human-readable, url-like names and enables name aggregation. Name aggregation improves the scalability by reducing the number of entries in the routing table, but it makes content multihoming difficult. However, multihoming plays a key role in improving the content delivery efficiency by selecting the best delivery paths from request to all the potential copies. The reason is hierarchical naming with aggregation indicates the correlation between content names and underlying network topology. To verify a content object in hierarchical naming scheme, a requester needs to obtain the public key of the publisher and there are various techniques available such as search engine and so on. But there must be a global public key infrastructure (PKI) so that ICN can bind content names to the keys. The prerequisite on PKI is considered as a major disadvantage of hierarchical naming.

On the other hand, flat naming, or the better-known name as self-certifying naming, is PKI independent. Namely, the key is bound to content name itself. Self-certification is usually achieved by either simply embedding the content hash into its name, or embedding both publisher's public key and content hash signed with secret key into the name. With either

way, self-certifying name is not human-readable, cannot be aggregated and but is able to free an ICN system from using a global PKI or trusting any other third parities. As claimed in [23], self-certifying name makes the infrastructure immune to denial-of-service attack without the need of understanding user trust model. Being PKI-independent is a big architectural advantage, and is often the reason why self-certifying naming is considered as a key enabler of the paradigm shift by its advocates. Last, instead of name aggregation, routing aggregation is feasible by taking advantage of some embedded structures in self-certifying names.

2.3 Name Resolution & Routing

There are two key functions in a conventional point-to-point network – *name resolution function* which finds out all potential locations of the requested content, and *routing function* which selects the best path based on some well-defined metrics. Since most ICN designs embrace the concept of locator/identifier split from pub/sub paradigm, content can be retrieved by merely using its identifier regardless of the actual location in a network. Therefore in ICN, to discover and retrieve a content for a given name, the network is expected to implement either name resolution which maps identifier to locator, or content-based routing which directly routes a request on its name.

The choice on these two approaches reflects on the tradeoff between efficiency and accuracy. On one hand, name resolution guarantees the content discovery, but may degrade delivery efficiency due to the query overheads. On the other hand, directly routing on content names is more efficient by bypassing the query steps but only promises probabilistic discovery, which positively correlates to the portion of the network being explored. From network traffic and operation complexity perspective, content-based routing is more expensive because it usually requires flooding the messages over the whole network to propagate the update. However, name resolution needs maintain at least two databases at a logically central point: identifier to locator mapping and reachability information, which renders the whole system vulnerable and inevitably increases management complexity.

In reality, NetInf and PSIRP use name resolution approach while DONA and CCN choose content-based routing. The implementation of routing heavily relies on the actual naming scheme adopted in the system and can be divided into two main parts: intra- and inter-domain routing. Intra-domain routing is usually implemented in various ways and differs mechanistically from one proposal to another. For inter-domain routing, some designs [1]

leverage the existing system like BGP, and some [2] implement their own protocols.

Technically, given a flat naming scheme, we can sidestep the quandary over identifier and locator with greedy routing. The application of greedy routing in ICN is studied in [25, 26] and shown to be a promising solution to mobility and other related issues like real-time content dissemination. Essentially, greedy routing implements a DHT underlay and transforms all the content routers into rendezvous points. Greedy routing can coexist with standard routing protocol in an ICN system as supplement for some specific settings (e.g. mobility). The successful application of greedy routing largely depends on how to find a proper greed embedding for a given network.

2.4 In-Network Caching

Caching is inherent in ICN architecture. In-network caching strategy is designed to alleviate the pressure on bandwidth and improve delivery efficiency by taking advantage of universally deployed in-network caches, and is reportedly the hottest research area in ICN. Content can be cached along the path from the data sink to source, which is referred as en-route caching. Its close resemblance in web caching is hierarchical caching. Or objects can be cached off-path in which case cooperation is needed to locate the nearest copy. In addition to content caching, some ICN designs can also cache requests, which is referred as request aggregation. The future requests on the same recently requested content can be suppressed before the content is actually cached by the router, so that extra network traffic can be avoided.

In ICN research, both centralized and distributed caching algorithms are studied. The centralized solutions are usually used to demonstrate an algorithm's overall behaviors or analyze its performance bounds. The distributed ones gain broader research interest due to its practical use. In distributed caching strategy design, a strategy is often decomposed into three parts and studied separately: admission policy, replacement/eviction policy and cooperation policy. The admission and replacement policy closely relate to the conventional caching algorithms which capture the content locality in a request stream. The cooperation policy is far more complicated not only because it needs to take both content and topology into account, but also because cooperation can be achieved in various ways.

Difference to the Prior Technologies

Though similar at its first glance, in-network caching is fundamentally different to the previous technologies for content distribution such as web,

Content Distribution Network (CDN) and Peer-to-Peer (P2P) caching.

Openness: ICN provides an open and unified caching framework with consistent naming, therefore is application-independent and makes content names network-aware. On the contrary, cache systems in the prior technologies are often closed, application-dependent and typically use proprietary protocols. Despite of some efforts in building a shared cache infrastructure [27], cache transparency is generally difficult to achieve with conventional solutions.

Design goals: Prior technologies are usually specifically designed for certain content and traffic type, whereas ICN is expected to deal with a wide range of content and traffic from web, streaming, file-sharing and etc. Specific design context leads to explicit design goals, e.g. P2P cache aims at reducing the traffic while web cache aims at reducing both traffic and latency. ICN cache needs to balance the competition among various traffic types on the limited space and operate at line-speed [28–30].

Chunk popularity: Content in ICN is sliced into small self-certifiable chunks for the efficiency of transmitting and caching. Chunking operation makes us reexamine those well-established theories and partial caching algorithms from web caching and P2P [31–33], since chunk-level popularity depends on in-file access pattern therefore can not be simply extrapolated from file popularity. Evidence [34] clearly showed that in-file access pattern can be an arbitrary mix of non-continuous portions. So-called *independent reference model* is also invalidated to some extent because chunks from the same file are often correlated. While [35] claimed none analytical work on chunk-level popularity has been done based on their knowledge, Chapter 8 of this thesis made its original contribution to this challenge.

Network topology: Conventional cache system usually resides in a network of regular structures like line cascading structure or hierarchical tree structure. Besides the cache location is predetermined, the access pattern is often known as given input (e.g. in CDN). Early work attests mathematical modeling and optimization are relatively straightforward in this setting [36–38]. However, moving to a more general graph makes mathematical techniques more difficult to apply. Most recent analytical work [39–43] on in-network caching is still based on hierarchical structures, while some [44, 45] attempt to advance to more general graphs by using approximate models.

High dynamic: Besides the diversity in content and traffic, the objects in cache are volatile and the demands may change frequently due to nodes' mobility. Classic technique developed for optimal placement problem is difficult to apply in such high dynamic setting, especially when the

apriori knowledge on topology and demands is missing [46]. Large-scale cooperation is often too expensive to achieve.

As we see, all aforementioned features distinguish in-network caching from the conventional caching technologies, further change the way we model, optimize and evaluate a caching algorithm. Especially cooperation policy, which plays a central role and interconnects all other features, attracts most research interest. In the following, we focus on cooperation policy of in-network caching strategy design, and present two different viewpoints on this topic.

Cooperative Caching

"Cooperation", "collaboration" and "coordination", all three terms are used in ICN literature, we stick to the first two and use both interchangeably in this thesis simply for the purpose of consistency. Cooperation in caching is meant for improving the visibility of cached content in order to achieve two main objectives: reduce content duplicates in the network to increase cache utilization (e.g. DHT and hierarchical caching); push popular content to the network edge to reduce latency (e.g. CDN and web caching). As [47] shows, the two objectives can become conflicting on the Pareto frontier. Comparing to the prior technologies, cooperation in in-network caching needs to be low-complexity, topology-aware and adaptive to the high traffic dynamic.

Modelling cooperation in a simple yet expressive way is not trivial, [47] describes a cooperation policy from two perspectives: search radius and tolerance on duplicates. Search radius represents the neighborhood size that cooperation can cover, while tolerance on duplicates represents how many duplicates are allowed in the neighborhood. Given system performance is optimal, the tolerance on duplicates is a dependent variable on search radius. Based on search radius, in-network caching strategies can be roughly categorized as bellow: en-route caching with none or limited cooperation [1, 3, 36, 38, 48–53], neighborhood cooperation [54–59] and global cooperation [43, 46, 47].

Most research activities are devoted to en-route caching strategies, because the requirements on line-speed operation and acceptable traffic overheads largely constrain the search radius of cooperation policy. The complexity of cooperation grows accordingly as we are pursuing higher and higher hit rate. Chapter 9 carefully studied the growth of traffic overhead on different graphs and its functional relation with search radius. Global cooperation, on the other hand, is seldom used in practice except for illustrating overall system behaviors.

Pessimistic View on Cooperation

Even though caching per se has long been recognized as an effective technique to improve scalability and efficiency, the viewpoints on cooperative caching are clearly divided into two opposing groups in ICN community. On one hand, the prior experience in web caching, especially in hierarchical caching, shows edge caches contribute most of the benefits [60]. The cache utilization drastically decreases in upstream caches. Especially in the network core, the dynamic is so high that it becomes extremely difficult for the core caches to capture the temporal/spatial locality in the request stream. Some believe that any attempt in deploying caches in network core or incorporating cooperative caching will only increase deployment complexity and even impose negative impacts on system performance.

The pessimistic view was supported by the study on large-scale web trace in [60], and is even exacerbated by the recent work done in Facebook [61] which shows the Facebook's internal caches only provide marginal benefits in their image-serving system. [31, 62] further examined the content popularity distribution in both web and P2P system, the results are even more unfriendly to cooperative caching, showing the cache utilization increases logarithmically with cache size after entering into the long tail of the content set. [4] further claims the ineffectiveness of cooperative caching pertains to all heavy-tailed popularity distributions. Therefore, recent work [63–65] proposes that ICN should be deployed incrementally at edge caches with existing technologies (e.g. HTTP, DNS and etc.).

Optimistic View on Cooperation

On the other hand, there is also plenty of supportive work for cooperative caching in both conventional context (web, P2P and share file system) [66–71] and ICN context [46, 47, 50, 54, 59, 72–76]. In general, the optimistic advocates of cooperative caching disagree with the opposing view in the following aspects:

1. Almost all the negative results are obtained by applying classic caching algorithms (LRU, LFU and etc.) which aims at maximizing local caching performance. However, in the context of cache networks, the immediate question is whether it is safe (or even correct) to assume local optimum indicates global optimum. On a nontrivial topology, the answer is clearly negative from optimization theory perspective.

2. Furthermore, filtering effect was already noticed in hierarchical caching systems. The thorough studies in [77, 78] confirmed that the effec-

tiveness of upstream caches is deteriorated by filtering effect, which is the direct consequence of optimizing locally without cooperation. Their work also showed the filtering effect can be ameliorated with a mixture of different caching algorithms. [72] further showed naive cooperation in en-route caching can effectively reduce the negative impact on cache utilization.

3. Cooperative caching emphasizes the fundamental difference between conventional caching and ICN caching, wherein network topology comes into play in performance measurement [30], parameter tuning, and protocol design [79]. [80] shows cache size should be well-tuned according to a node's degree centrality. Chapter 9 of this thesis shows the overhead due to cooperation depends on a network topological properties. However, thorough study on the effects of network topology in ICN context is still in a vacuum state based on our knowledge.

4. "Zipf distribution matters", as argued in [81] which holds an negative view on in-network caching, indicates only the most popular content resides at the head part of a popularity distribution determines the actual caching performance. Both [81] and [4] use this as a strong argument against (cooperative) in-network caching. However, it must be pointed out that the aggregated cache size, especially its ratio to the aggregated size of the popular content, as an important determinant factor of caching performance, was unfortunately missed in the arguments. Noticing one major goal of cooperative caching is aggregating the networked caches as a single cache, arguments in [4, 81] may need further investigation by explicitly taking aforementioned ratio and cooperation into account.

5. The shift from single cache to cache network requires better understanding of their commonalities and fundamental differences. It also suggests new measurement metrics and evaluation methods which are either under development or largely missing in the current research. Without a sound set of criteria, it is too early and hasty to sentence cooperative caching to its death. [47] initiated the first discussion on thorough evaluation of cooperation policy in a distributed caching strategy.

In general, the quality of an ICN design is evaluated from various aspects such as scalability, security, complexity and so on. From caching perspective, a system is usually measured with hit rate. This widely-used metric has a history as long as that of computers. In ICN context, pursuing

higher hit rate is equivalent to pursuing an algorithm which can utilize the aggregated cache size in the network as a single big cache. However, from measurement perspective, hit rate contains limited information to describe a system and it is arbitrary to judge the quality of a design merely with hit rate. As argued in Chapter 3, other metrics like footprint reduction, which measures the traffic within a network, should also be adopted in evaluation. [47] shows after hitting the Pareto boundary of system performance, higher hit rate is achieved at the price of lower footprint reduction. Nonetheless, the clear tradeoff between intra- and inter-domain traffic in caching strategy design does not seem to capture enough attention in most of the prior work, let alone the cooperation policy which plays a key role in balancing these two metrics.

2.5 Discussion

Though it has been over a decade since the concept of ICN was proposed, there are many open questions yet to be answered and requires better understanding. Some of them should be categorized as design choices that cannot be answered with a simple "yes" or "no", but mostly depends on the specific context and various tradeoff taking different factors into account. Whereas some are important system architectural problems that need further investigation since many researchers hold diametrically opposing views upon them.

In this thesis, we try to hold an neutral stance on the viewpoints towards in-network caching, more specifically, cooperative in-network caching. Though our work give supportive results on the effectiveness of cooperation in improving caching performance, we feel that it is still too early to jump to any hasty conclusion. Instead, we focus on the interactions among the content, topology and cooperation policy, and try to study how the cooperation takes effects in various settings from an impartial viewpoint. In other words, given the cooperation is implemented in an ICN system, what is its role in a distributed caching algorithm and how it should be designed to satisfy certain well-defined requirements.

Besides the aforementioned neutral stance, the ICN under our discussion is a simplified model which remains the core design principles and possesses most commonalities in different proposals. We know the actual implementation varies from design to design, this thesis only focuses on the general architecture instead of delving into low-level mechanistic details. Sometimes we use a specific ICN design (e.g. CCN) as an example, but we try to keep our claims as general as possible.

Table 2.1: Comparison of four important ICN designs [17]

	DONA	NetInf	PSIRP	CCN
Published	2007	2008	2008	2009
Project		SAIL	PURSUIT	NDN
Operation primitives	`REGISTER`, `FIND`	`GET`	`PUBLISH`, `SUBSCRIBE`	`REGISTER`, `INTEREST`
Namespace	Flat	Flat	Flat	Hierarchical
Content integrity	Signature, PKI independent	Signature or content hash, PKI independent	Signature, PKI independent	Signature, external trust source
Human-readable names	No	No	No	Possible
Content granularity	Objects	Objects	Objects	Packets
Routing aggregation	Publisher/explicit	Publisher	Scope/explicit	Publisher
Routing request	Name-based (via RHs)	Hybrid NRS and name-based	NRS (rendezvous)	Name-based
Routing response	Reverse request path or direct IP connection	Reverse request path or direct IP connection	Source routing using Bloom filter	Reverse request path using router state
API	Synchronous get	Synchronous get	Publish/Subscribe	Synchronous get
Transport	IP	Many including IP	IP/PSIRP	Many including IP

Chapter 3

Measurement Metrics and Methodology

Information-centric networks are an interesting new paradigm for distributing content on the Internet. They bring up many research challenges, such as addressing content by name, securing content, and wide-spread caching of content. Caching has caught a lot of attention in the information-centric networks research community, but a lot of the work suffers from a poor understanding of the different metrics with which caching performance can be measured. In this chapter we present a comprehensive overview of different caching metrics that have been proposed for information-centric networks, and discuss their merits. As we show, many commonly used metrics have several failure modes which are largely ignored in literature. We identify these problems and propose remedies and new metrics to address these failures. Our work highlights the fundamental differences between information-centric caches and "traditional" cache networks and we demonstrate the need for a systematic understanding of the metrics for information-centric caching. We also discuss how experimental work should be done when evaluating networks of caches. In addition, we give a detailed description on the design of Litelab – the experiment platform we used in the evaluation.

3.1 Introduction

Information-centric networking (ICN) provides a new paradigm for addressing and accessing content on the Internet. The current Internet was developed as a host-centric network, where the main focus was on interconnecting computers, or hosts, but the modern usage of Internet is very much information-centric, i.e., users do not care from where the information they

want to access comes from; they simply are interested in getting the information. The web is in its essence a host-centric system, although content delivery networks (CDN) and technologies do break the dependence on (particular) hosts serving specific content to some extent. However, fundamentally the web is still a host-centric system and its different components, such as naming and security, are tied to this host-centric world.

ICN puts the focus on the content as opposed to the hosts to address the architectural issues preventing the web from becoming a full-blown information-centric system. There are several independent proposals around ICN [1–3, 9]. They each present a different solution to try to re-construct the current Internet, and build a new architecture around the notion of content. While the details in the proposal differ, we can identify three common components that are fundamental to information-centric networking: addressing content by name, securing content, and wide-spread caching.

Of these three, the last one, wide-spread caching, seems to have attracted the most attention in the research world lately. Our focus in this chapter is on measuring the effectiveness of caching, but first we outline the main issues in naming and security, highlighting in particular how they impact caching.

Addressing content by name is an important change to how content is addressed in the web. Although URLs are "names" of content, they have internal structure which indicates the server hosting that content as well as a local "path" on the server to the content. While at first sight similar, names in ICN may have structure, but the structure does not identify a particular server in the network that would need to be accessed to retrieve the content. Content discovery is a big challenge in ICN and two different choices seem to emerge from the ICN proposals. One possibility is to use an indexing service [9] which keeps track of copies of objects; however it is not clear if this will scale up to a global scale. The other possibility, used in most of the other ICN proposals, is to route requests based on some components in the content name, with the hope that this routing converges on a copy of the object. Especially in the latter case, en-route caching becomes an attractive option to speed up discovery and spread the load on content distribution. In this chapter, we mainly follow this kind of a model and assume that content requests are routed in the same way from all over the network and that the routing converges towards existing copies of the content. Caching is assumed to happen en-route and cached copies are not tracked in any way.

Security on the web is essentially based on identifying the server pro-

viding the content via SSL and its associated certificates. Since content no longer has a single origin in ICN, this approach does not work anymore. Instead, the ICN approaches all focus on securing the content, by ensuring via signatures and public keys that the content has not been tampered with in the network. This also allows caching to take place since any piece of content, no matter from which server it is served from, can be authenticated to be the same content that the originator put in the network. Obviously, this does not tie the content to a real world entity, but this can be achieved in a similar way to how it is done on the web.

Finally, wide-spread caching is used to store the content in the network and allow for faster delivery. Caching also reduces traffic in the network and is therefore attractive for network operators since it has the potential to reduce their costs.

Although all three above factors are fundamental to ICN, caching seems to have attracted the most attention in recent research. Caching is a topic that has been researched in many different contexts and it is attractive in the sense that it can be measured quantitatively with relative ease, whereas effectiveness of naming schemes or security solutions tend towards more qualitative measurements. However, a lot of the work on caching in ICN uses the "old" caching metrics that are known from processors or web caches. As we discuss in this chapter, ICN is a network of caches and there are fundamental differences between ICN caching and, e.g., web caching. Web caches can also be organized in networks, however they work in a fundamentally different way from the network of caches in ICN.

Our key contributions in this chapter are to highlight the fundamental differences between different caching metrics and show how they impact the metrics that should be used to measure the effectiveness of caching. We present several metrics and show how they vary in their complexity and expressiveness. The goal of this chapter is to demonstrate that caching in ICN is a novel area of research and that existing solutions have only limited applicability in this field.

3.2 System View of Cache Networks

In this chapter we focus on CCN-like [1] ICN where requests for (pieces of) content are forwarded via routers and these routers are equipped with a cache where they can store content. We focus on the case of a single ISP, as shown in Figure 3.1, which depicts several clients, one server, and a network of routers. Some of the routers are connected towards clients and some towards servers. We do not distinguish whether these are actual

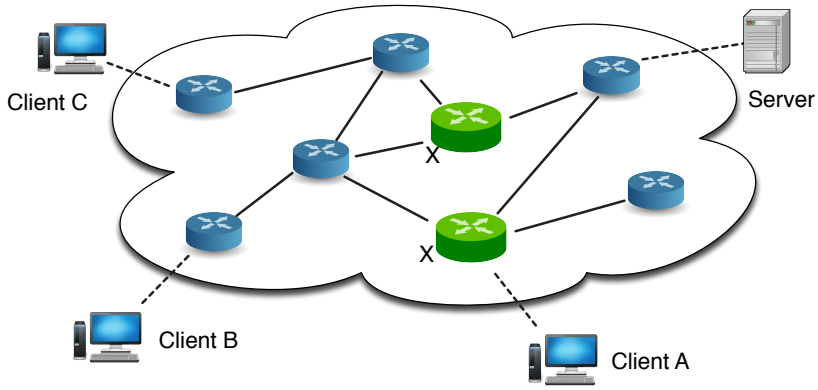


Figure 3.1: Model of the network

clients or other ISPs that are connected to the clients; they represent the incoming requests. Likewise, the servers represent sources of content and do not necessarily need to be connected to this particular ISP. From the ISP's point of view, the traffic reduction that caching brings can have two goals. Firstly, it reduces traffic towards the servers (*inter-ISP traffic*) which typically has a (direct) financial cost for the ISP. Secondly, it reduces traffic within the ISP's own network (*intra-ISP traffic*). Intra-ISP traffic has no direct connection to the ISP's costs, however lower intra-ISP traffic means that the ISP is able to serve more customers with the same infrastructure, which does have a positive financial effect.

The fundamental difference between ICN caching and, for example, web caching is that in ICN caching we have a *network of caches* that can work together to optimize the performance of the whole network. Although hierarchical web caching implemented similar networks of caches, each cache was operated by a different real-world entity and attempted to optimize its own performance. This led to issues like the filtering effect [60, 77] where first caches in the hierarchy capture the most popular objects because they attempt to optimize their own performance. This in turn leads to the following caches to see a request stream with less locality, making their performance suffer. In the context of web caching where every cache is operated by a different entity, this is reasonable, but in the context of ICN, a single entity controls multiple caches and is able to make them cooperate to optimize the overall performance. Results in [72] show that even a simple randomization of where to cache a particular piece of content has a significant boost of overall performance because it mitigates the filtering effect. As example, consider Figure 3.1. In the web caching model, the routers

next to the clients would cache the most popular content, but in ICN, it is feasible to have some other routers do that, for example the green routers marked with X. In other words, [72] shows that caches being less greedy in optimizing their own performance is beneficial to the whole system. This implies that when evaluating networks of ICN caches, *we need to look at the performance of the whole network* instead of optimizing performance of individual caches.

3.3 Core Metrics and Potential Extensions

We now present the main contribution of the chapter and outline different metrics that can be used to measure performance of a network of caches. We consider three metrics that have been used in literature and present a new metric called *coupling factor*.

(Byte) Hit Rate

Typically cache performance has been measured via hit rate, which captures the ratio between cache hits (requests found in the cache) to the total number of requests seen by the cache. Byte hit rate is its natural extension where every hit is weighted by the size of the object, hence byte hit rate measures the reduction in outgoing traffic from the cache. As our focus is on traffic reduction, we use byte hit rate in the following. When we apply byte hit rate as the performance metric, we effectively aggregate the whole network of caches as a single cache and look at its performance. (Note that since multiple caches may hold a copy of the same object in ICN, such an "aggregate" cache has less storage than the individual caches together; this does not influence the metric.) Byte hit rate is location-agnostic since it only cares whether there was a hit in any cache; it does not provide any information about where the hit happened.

Byte hit rate is an often-used metric, partly because caches have traditionally been measured by hit rate, partly because it is easy to compute, and partly because it translates directly to savings in inter-ISP traffic, i.e., financial savings. Reducing duplicate copies of objects in the network is the most effective way of improving byte hit rate; however an efficient reduction in number of copies requires an efficient cooperation method between the caches to discover the cached copies.

While byte hit rate is easy to compute, it treats the network of ICN caches as a black box since it does not take into account where the hit happens. Another argument against byte hit rate as a metric stems from the current trends of content distribution in the Internet. Large content

delivery networks or content providers, who host the most popular content, install their servers in or close to the ISPs where the users are. Although the servers are in the ISP's network, the normal way of calculating byte hit rate would consider them external, thus traffic to them would be counted the same way as any outgoing traffic; yet there is typically no cost to the ISP for traffic to them. Hence, most of the popular content actually comes from inside the ISP, and only the savings in the less popular content are relevant for the ISP. Byte hit rate is not able to capture this and therefore we recommend that it should not be used as a general metric; in specific situations it may be appropriate, but it is not appropriate as a general metric for all situations.

Average Hops

Measuring the number of hops a request needs to traverse in order to find the content is also a metric that has been recently used [82]. It is appealing in the sense that it augments byte hit rate by taking into account where the hit happens, however it does not provide any meaningful way of estimating savings in outgoing traffic. In addition, as is done in [82], average hops is sometimes used as a proxy for download latency.

Our previous work [72] shows that average hops as a metric does not discriminate well, i.e., while the qualitative ranking of caching solutions is correct, the quantitative differences between them are very small, which can easily lead to an impression that the performance differences would be small [82]. Other metrics we consider in this chapter do not share this weakness. The reason behind this is that average hops measures absolute values and because many networks are scale-free, the number of hops is typically small. Hence, differences between caching strategies will appear small, but this is actually an artifact of the metric, not an indicator that the strategies would be close to each other according to other metrics.

Another difficulty in using average hops as a metric relates to what value to assign to content retrieved from outside the ISP, i.e., a miss. Assigning a high value puts emphasis on avoiding misses, i.e., the metric becomes similar to hit rate. Assigning a low value emphasizes the location of content in the ISP's network, i.e., it gives an impression of intra-ISP traffic. However, as the amount of data is not part of the metric, it does a poor job in capturing something useful and a better metric, like footprint reduction described below, is needed.

Footprint Reduction

Traffic footprint in traffic engineering is defined as a product traffic volume and the distance it travels within the network. The distance is usually measured in terms of hops. To calculate the footprint, we need to sum up all the products of every data packet sizes and their travel distances. Footprint reduction is the fraction of reduction in footprint when caching is enabled.

Compared to byte hit rate, footprint reduction takes into account *where* the hit happens, since the number of hops is counted in the metric. However, since footprint reduction uses the size of the content, it gives more accurate information about traffic reduction than simply using the average hops. Also, it measures relative change and gives therefore a better picture of the differences between caching strategies. Note that footprint reduction measures only reduction of intra-ISP traffic and does not give any indication about possible reductions in inter-ISP traffic.

Byte hit rate and footprint reduction are the two key metrics in evaluating performance of networks of ICN caches, but they must be used in conjunction; using only one of them leads to biased results (using only average hops will lead to even more bias). For example, consider two caching strategies which achieve the same byte hit rate, but different footprint reductions. Higher footprint reduction indicates that the hits happen closer to the clients, thus less intra-ISP traffic and generally better performance for the users. We have identified and quantified the tradeoff between byte hit rate and footprint reduction [47] and will briefly outline this tradeoff below.

A naive solution for improving footprint reduction is to place the popular content as close to the clients as possible, i.e, edge caching [82]. However, this leads to large redundancy in cached content and leads to (much) lower byte hit rate. This tradeoff between the two metrics is mediated by a *cooperation policy* which enables richer cooperation between the clients than a simple en-route caching allows [47]. As discussed in [47], the tradeoff can be mediated by adjusting the number of copies for a content item and the range of how widely we search for the content in the network in case of a miss. The search range covers all possible cases from en-route caching to searching the whole network (obviously with a cost that would need to be accounted for). Adjusting the number of copies is harder to do exactly, but simple mechanisms like Cachedbit [72] are likely to be sufficient in many cases. For more details, we refer the reader to [47, 72].

One thing is worth pointing out is "no caching" was selected as baseline in calculating footprint reduction. In other words, it represents the traffic

saving comparing to a special caching strategy in which all the caches are disabled. Doubts have been raised that if we change to another caching strategy as baseline, whether the results (or observation) will remain valid. Below, we give a simple proof to show it actually does not matter which strategy we choose as a baseline, the results are affine.

Let x_α, x_β and x_θ denote the traffic footprint for caching strategy α, caching strategy β, and no caching θ respectively, and y_α, y_β, and y_θ are the corresponding footprint reduction. Our current way of calculating footprint reduction is defined as:

$$y_\alpha = 1 - \frac{x_\alpha}{x_\theta} \tag{3.1}$$

$$y_\beta = 1 - \frac{x_\beta}{x_\theta} \tag{3.2}$$

$$y_\theta = 1 - \frac{x_\theta}{x_\theta} = 0 \quad \text{(baseline)} \tag{3.3}$$

As we can see, x_θ is just the baseline we are comparing to. Obviously, the footprint reduction y_θ is zero when comparing against itself. We can of course change the baseline to the caching strategy β's footprint x_β, then we have the corresponding new metrics y'_α, y'_β and y'_θ calculated as below:

$$y'_\alpha = 1 - \frac{x_\alpha}{x_\beta} \tag{3.4}$$

$$y'_\beta = 1 - \frac{x_\beta}{x_\beta} = 0 \quad \text{(baseline)} \tag{3.5}$$

$$y'_\theta = 1 - \frac{x_\theta}{x_\beta} \tag{3.6}$$

From Eq (3.2), we have $x_\beta = x_\theta(1 - y_\beta)$. If we let $a = 1 - y_\beta$ and $b = 1 - \frac{1}{1-y_\beta}$, then Eq (3.4), (3.5) and (3.6) can be rewritten as follows

$$y'_\alpha = 1 - \frac{x_\alpha}{x_\beta} = 1 - \frac{x_\alpha}{x_\theta(1-y_\beta)} = \frac{1}{1-y_\beta}(1 - \frac{x_\alpha}{x_\theta}) + 1 - \frac{1}{1-y_\beta} = ay_\alpha + b \tag{3.7}$$

$$y'_\beta = 1 - \frac{x_\beta}{x_\beta} = 1 - \frac{x_\beta}{x_\theta(1-y_\beta)} = \frac{1}{1-y_\beta}(1 - \frac{x_\beta}{x_\theta}) + 1 - \frac{1}{1-y_\beta} = ay_\beta + b \tag{3.8}$$

$$y'_\theta = 1 - \frac{x_\theta}{x_\beta} = 1 - \frac{x_\theta}{x_\theta(1-y_\beta)} = \frac{1}{1-y_\beta}(1 - \frac{x_\theta}{x_\theta}) + 1 - \frac{1}{1-y_\beta} = ay_\theta + b \tag{3.9}$$

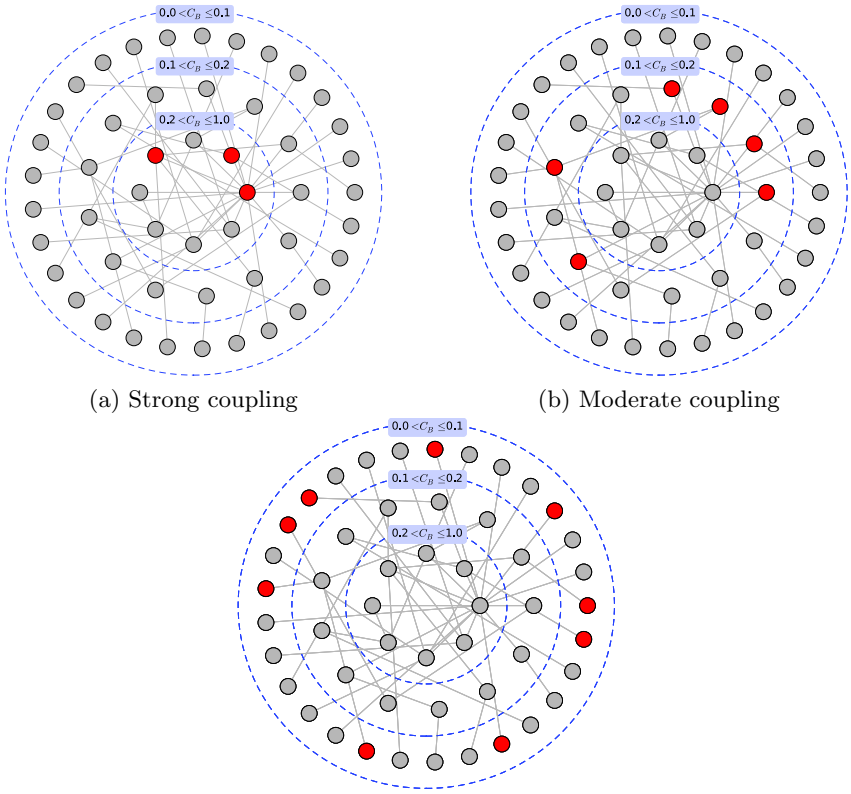
So we can see that the new metrics (y'_α, y'_β and y'_θ) are simply affine transformation of the old ones (y_α, y_β and y_θ). It means the footprint reduction is independent of the choice on which caching strategy as baseline, since it will not change the "ranking" of the results. In some sense, geometrically, it simply means where we want to set our "origin" point, namely "0" point.

Coupling Factor

We propose a new metric, the *coupling factor*, to capture the effects of the network topology on the performance of caching. We achieve this by identifying the "position" in the network where the hit happens. Recall that byte hit rate does not give any information about where the hit happens, and footprint reduction is limited to finding content only along the routing path. A cooperation policy that searches wider in the network is able to find content in other locations as well. In this case, the position has a direct impact on the calculation of the metrics.

We define the coupling factor as a function of content popularity and network topology and it measures the impact of topology on content placement, and thus the impact on metrics like byte hit rate and footprint reduction. Content popularity is easy to obtain, but for characterizing topology, we have many more options, such as degree centrality, betweenness centrality, closeness centrality and so on. Therefore, coupling factor can have several forms depending on which metrics are used in calculating the correlation, but the general idea is the same: we need a way of showing the relationship between popularity and topology.

Figure 3.2 shows how different degrees of coupling affect the placement of the most popular content. The red dots represent the most popular content and the concentric circles group nodes according to their betweenness centrality. Strong coupling means that the most popular content is placed in the nodes with high betweenness, i.e., the network core. Weak coupling means the opposite, i.e., the popular content is placed at the network edge. (Strictly speaking, if using correlation between popularity and node degree as a metric, strong coupling is indicative of strong positive correlation and weak coupling implies strong negative correlation.) By adjusting the range of the search and the number of copies, we can influence the placement of content in the system, i.e., adjust the degree of coupling. When the popular content is in the core, we improve byte hit rate and when the popular content is near the edge, we favor footprint reduction. This means that the two parameters, search range and number of copies, can be used to adjust the tradeoff between the two metrics.



(a) Strong coupling

(b) Moderate coupling

(c) Weak coupling

Figure 3.2: Coupling between content popularity and network topology

Other Useful Metrics

The metrics we presented in Section 3.3 are by no means the only metrics that can be used for measuring cache performance.

As is well known, the popularity distribution significantly influences caching performance in all kinds of caches. However, new content is constantly added and popularity of content changes which may influence the metrics that are being used to measure the performance of caches. Typically, some kind of aging is used to rid the system of old popularity information and test a caching strategy's ability to adapt to changes in content popularity. Since popularity typically follows a power law and is characterized by a (single) parameter, it is a common technique to measure the effects of changes in that parameter on other system metrics.

Another interesting measure would be to find a way to quantify the filtering effects. Filtering effect has been noticed in hierarchical cache sys-

tems [77], and it refers to the phenomenon that the popularity of the most popular part in a miss stream is "flattened" because the downstream router filters out the most popular content. Filtering effect degrades the caching performance of upstream routers. We do not yet have a good metric to quantify the filtering effect, however results in [72] seem to indicate that simple solutions might be able to fight the filtering effect, thus obviating the need for its measurement. However, this is a question for future research.

As mentioned, content popularity changes dynamically, but also the network topology changes, be it due to failures or simply an evolution of the infrastructure. This question needs more research in order to evaluate how often these kinds of topology changes affect an information-centric cache network and how large the impact would be.

3.4 Beyond the Metrics: Experiment Design

Choosing the right metrics is the first step, but it is not enough. Designing experiments is as critical as selecting the right measurements, since it directly influences the ability to extract the right information from the experiment, as needed by the metrics that are to be computed. A poorly designed experiment will not allow the correct information to be extracted, leading to possibly erroneous conclusions. In terms of ICN experiments, there are three key elements in experiment design: *content*, *topology*, and *traffic model*.

Content popularity is a key factor affecting performance of caching systems. In the absence of publicly available request traces from recent years, many researchers are forced to use synthetic request traces. For synthetic traces, it is important that its characteristics match those of realistic traces as closely as possible; obviously it will not be an exact match, but basic statistical characteristics should match real traces. Research has shown that real world traces can often be modeled by either Zipf of Weibull distributions [83]. A further aspect of modeling the content is the size of the content set, i.e., how many objects are included in the synthetic trace; a real trace has its own fixed number of objects. The reason why the content set size is important is the "heavy tail" nature of these popularity models which drags down performance is the content set is too large (for the amount of cache storage in the system).

Topology of the network is also of high importance in a correct experimental setting. Research has showed that most realistic networks, including the Internet are scale-free, so a Barabási-Albert model could be used to generate synthetic network topologies. However, as work in [84] shows, real

ISP router topologies do not always conform to the scale-free model, so it is also important to experiment with real topologies. As opposed to request traces, network topologies, both at ISP and AS level, are readily available, so unless the experiment setting requires a network much larger than exists in traces, using a real network topology is preferable to synthetic topologies. Obviously, work needs to consider many different topologies in order to eliminate specifics of particular topologies from the results.

The third element, traffic, has two components: how much traffic and between which points it goes? In the real world, requests from users at the network edge and traffic volume is typically proportional to the population of an edge routers geographical location. These observations have led to the so-called "gravity model" which has been used in many papers. Typically servers are connected to routers with high degree, i.e., near the core of the network and clients are connected to edge routers. The question then becomes how many clients should the network have? If all routers are assigned as servers or clients, there is no "intermediate layer" of caches and this may lead to a stronger filtering effect which degrades the performance of the system. Based on our experience, we have observed that placing a small number of servers according to the gravity model and then assigning 20–30% of the edge routers as clients works well. In addition, the placement of clients should be randomized, varied from one experiment run to another, and we should perform a sufficient number of repetitions to guarantee statistically meaningful results, using for example confidence intervals to determine the number of repetitions.

3.5 Evaluation Platform: Litelab

For network researchers, large-scale experiment is an important tool to test distributed systems. A system may exhibit quite different characteristics in a complex network as opposed to behavior observed in small-scale experiments. Thorough experimental evaluation before real-life deployment is very useful in anticipating problems.

This means that the system should be tested with various parameter values, like different network topologies, bandwidths, link delays, loss rates and so on. It is also useful to see how the system interacts with other protocols, e.g. routers with different queueing policies.

Due to the large parameter space, researchers usually need to run thousands of experiments with different parameter combinations. As Eide pointed out in [85], replayability is critical in modern network experiments. Not being able to replay an experiment implies the results are not repro-

ducible, which makes it difficult to evaluate a system, because the results from different experiments are not comparable.

A simulator has been a popular option due to its simplicity and controllability. It also has other benefits like reproducible results and low resource requirements. However, a simulator is only as good as the models used. Choosing the right granularity of abstraction is a tradeoff between more realistic results and increased computational complexity.

Experiments on real systems can overcome many problems of simulators, because all the traffic flows through a real network with real-world behavior. However, running large-scale real-world experiments requires a lot of resources. Virtualization may help, but configuring and managing large experiments is still difficult.

Recently, high performance clusters are becoming common, virtualization technology advances, and overlay networks seem to become the de facto paradigm for modern distributed systems. All these emerging technologies change the way we build and evaluate networked systems.

In the following, we present *LiteLab*, our flexible platform for large-scale networking experiments. We show its design, functionalities, key features and an evaluation of its accuracy and performance. With LiteLab, researchers can easily construct complex network on a resource-limited infrastructure.

LiteLab is easy to configure and extend. Each router and links between them can be configured individually. New queueing policies, caching strategies and other network models can be added in without modifying existing code. The flexible design enables LiteLab to simulate both routers and end systems in the network. Researchers can easily plug in user application and study system behaviors. LiteLab takes advantages of overlay network techniques, providing a flexible experiment platform with many uses. It helps researchers reduce the experiment complexity and speeds up experiment life-cycle, and at the same time, provides satisfying accuracy.

General Architecture

The goal of LiteLab is to provide an easy to use, fully-fledged network experiment platform. Figure 3.3 shows the general system architecture. LiteLab consists of two subsystems: *Agent Subsystem* and *Overlay Subsystem*, presented below.

We first illustrate how LiteLab works by describing how an experiment is performed on this experiment platform.

All experiments are *jobs* in LiteLab and are defined by a job description archive provided by a user. An archive can contain multiple configuration

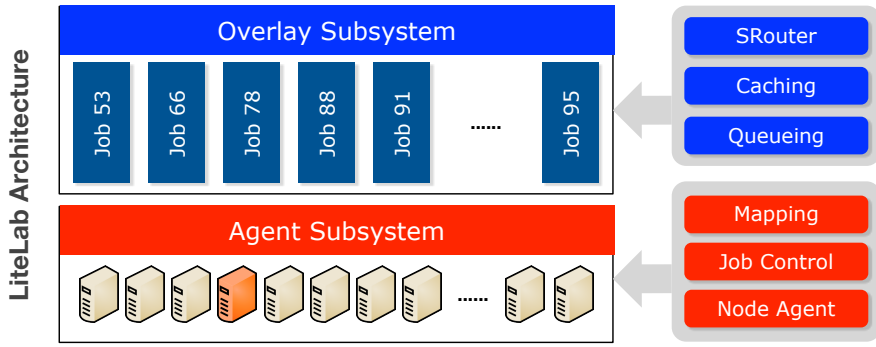


Figure 3.3: LiteLab Architecture

files which specify the details of the experiment, e.g., network topology, router configuration, link properties, etc. The Agent Subsystem has one leader node which is responsible for starting and managing jobs. We use the Bully election algorithm for selecting the leader dynamically.

Second, the user submits the job to LiteLab which processes the job description archive, determines needed resources and allocates necessary physical nodes from the available nodes. We have developed and run LiteLab on our department's cluster, but the design puts no constraints on where the nodes are located.[1] Nodes with lighter loads are preferred.

Third, LiteLab informs the selected nodes and deploys an instance of the Overlay Subsystem on them (see below). The Overlay Subsystem is started to construct the network specified in the job description archive.

Finally, LiteLab starts the experiment, and the job is saved into the *JobControl module*, which continuously monitors its state. If a node is overloaded, LiteLab will migrate some SRouters to other available nodes to reduce the load. If an experiment successfully finished, all the log files are automatically collected for post processing.

Agent Subsystem

The Agent Subsystem provides a stable and uniform experiment infrastructure. It hides the communication complexity, resource failures and other underlying details from the Overlay Subsystem. It is responsible for managing physical nodes, allocating resources, administrating jobs, monitoring experiments and collecting results. The main components of the Agent

[1] For geographically dispersed nodes, strong guarantees about network performance may be hard or impossible to provide.

Subsystem are NodeAgent, JobControl and Mapping.

1. **NodeAgent** represents a physical node, thus there is one-to-one mapping between the two. It has two major roles in LiteLab. First, it serves as the communication layer of the whole platform. Second, it presents itself as a reliable resource pool to Overlay Subsystem. We use the Bully algorithm to elect a leader responsible for managing the resources and jobs.

2. **JobControl** manages all the submitted jobs in LiteLab. After pre-processing, JobControl allocates the resources and splits a job into multiple tasks which are distributed to the selected nodes. The job is started and continuously monitored.

3. **Mapping** maps virtual resources to physical resources. The goal is to maximize resource utilization and perform the mapping quickly. It is also a key component to guarantee the accuracy. Mapping module runs an LP solver to achieve the goal.

Resource Allocation: Dynamic Migration

Resource allocation focuses on the mapping between virtual nodes and physical nodes, and it is the key to platform scalability. We have subdivided the resource allocation problem into two sub-problems: *mapping problem* (below) and *dynamic migration*

The mapping should not only maximize the resource utilization, but also guarantee there is no violation of physical capacity. We take four metrics into account as the constraints: CPU load, network traffic, memory usage and use of pseudo-terminal devices. Deployment of the software-simulated routers (SRouter) must respect the physical constraints while optimize the use of physical resources.

Suppose we have m physical nodes and n virtual nodes. We first construct an $m \times n$ deployment matrix D. All the elements in D have binary values. If $D_{i,j}$ is 1, then virtual node i is deployed on physical node j, otherwise $D_{i,j}$ is 0. We denote C_i as the CPU power, M_i as the memory capacity, U as egress bandwidth and V as ingress bandwidth of physical node i. We also denote c_j, m_j, u_j and v_j as virtual node j's requirements for CPU, memory, egress and ingress bandwidth respectively.

We model the processing capability of a node in terms of its CPU power:

$$\sum_{j=1}^{n} D_{i,j} \times c_j \leq C_i, \forall i \in \{1, 2, 3..m\} \tag{3.10}$$

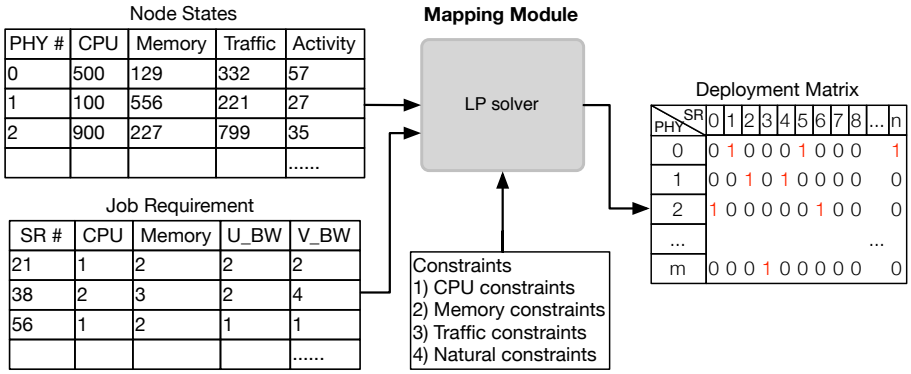


Figure 3.4: Inputs and output of Mapping module

Total memory requirements from virtual nodes running on the same machine should not exceed its physical memory:

$$\sum_{j=1}^{n} D_{i,j} \times m_j \leq M_i, \forall i \in \{1, 2, 3..m\} \tag{3.11}$$

The aggregated bandwidths are also subject to physical node's bandwidth limit:

$$\sum_{j=1}^{n} D_{i,j} \times u_j \leq U_i, \forall i \in \{1, 2, 3..m\} \tag{3.12}$$

$$\sum_{j=1}^{n} D_{i,j} \times v_j \leq V_i, \forall i \in \{1, 2, 3..m\} \tag{3.13}$$

A virtual node can only be deployed on one physical node, and the total number of virtual nodes is fixed. Two natural constraints follow:

$$\sum_{i=1}^{m} D_{i,j} = 1, \forall j \in \{1, 2, 3..n\} \tag{3.14}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} D_{i,j} = n \tag{3.15}$$

Our mapping strategy is to use as few physical nodes as possible, and give preference to less loaded nodes. In other words, we try to deploy as many virtual nodes as possible on physical nodes with the lightest load. We define node load L:

$$L = w_1 \times avg\_cpu\_load + w_2 \times traffic \\ +w_3 \times memory\_usage + w_4 \times user\_activities \tag{3.16}$$

The four metrics are given different weights to reflect different level of importance. In our case, we set $w_1 > w_2 > w_3 > w_4$, but this choice is rather arbitrary; Emulab uses a similar rationale [86]. In our evaluation, we have found that our simple rule for the weights is sufficient, but further study would be required to gain more understanding on their importance.

Larger L implies heavier load. We give each machine a preference factor p_i equal to the reciprocal of its load, L^{-1}. Node with the lightest load has the largest preference index.

We formalize the mapping problem into a *linear programming problem* (LP). The objective function is as follows:

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_i \times D_{i,j} \tag{3.17}$$

subject to the constraints in equations (3.10)–(3.15).

Each node sends its state information to the leader, which then has global knowledge needed for solving the LP problem. Our LP solver is a python module, which takes node states and job description as inputs, and outputs the optimal deployment matrix. Figure 3.4 shows how Mapping module works.

We also adopted other mechanisms into our LP solver to further improve the efficiency by reducing the problem complexity. We discuss these in detail in Section 3.5.

Resource Allocation: Dynamic Migration

The static mapping cannot efficiently handle the dynamics during an experiment. For example, a node overloaded by other users' activities may skew our experiment results. We use dynamic migration to solve these problems.

Dynamic migration is implemented as a sub-module in NodeAgent. It keeps monitoring the load (defined by e.q (3.16)) on its host. If NodeAgent detects a node is overloaded, some tasks will be moved onto other machines without restarting the experiments. Migration is not able to completely mask the effects from other users, but can alleviate the worst problems.

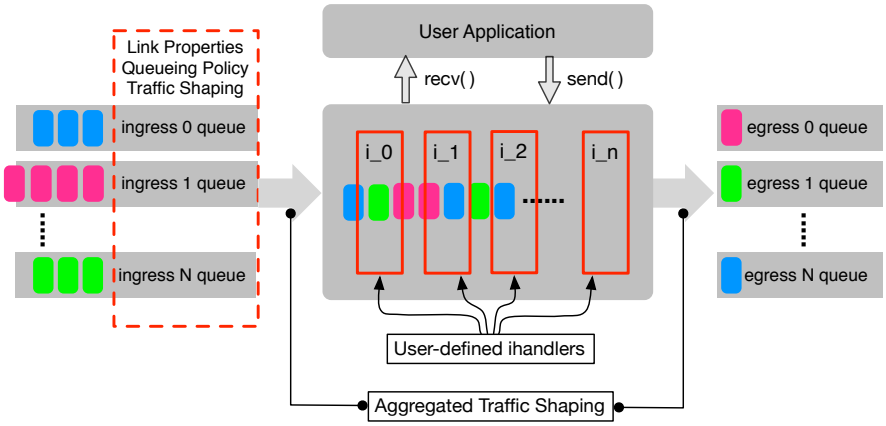


Figure 3.5: Logical structure of SRouter

Overlay Subsystem

Overlay Subsystem constructs an experiment overlay by using the resources from Agent Subsystem. One overlay instance corresponds to a job, therefore LiteLab can have multiple overlay instances running in parallel at the same time. JobControl module manages all the created overlays.

The most critical component in Overlay Subsystem is SRouter, which is a software abstraction of a realistic router. Due to its light-weight, multiple SRouters can run on one physical node. Users can configure many parameters of SRouters, e.g., link properties (delay, loss rate, bandwidth), queue size, queueing policy(Droptail, RED), and so on.

Queues

Figure 3.5 shows SRouter architecture and how the packets are processed inside. SRouter maintains a TCP connection to each of its neighbors. The connection represents a physical link in real-world. For each link, SRouter maintains a queue to buffer incoming packets. In the job description archive, user can specify the link delay, loss rate and bandwidth for each individual link. All these properties are modelled within the SRouter so that they are not subject to TCP dynamics.

SRouter maintains three FIFO queues inside: *iqueue*, *equeue* and *cqueue*. All incoming packets are pushed into *iqueue* before being processed by a chain of functions. All outgoing packets are pushed into *equeue*. Aggregated traffic shaping is done on these two queues. If a packet's destination is the current SRouter itself, then it enters into *cqueue*. Later, the packet

will be delivered to user application.

Processing Chain

We borrowed the concept of chains of rules from *iptables* when we designed SRouter. If a packet waiting in the ingress queue gets its chance to be processed, it will go through a chain of functions, each of which can modify the packet. We call such a function an *ihandler*. If an ihandler decides to drop the packet, then the packet will not be passed to the rest of the ihandlers in the chain.

The default ihandler is *bypass_handler*, and is always the last one in the chain. It simply inserts an incoming packet into *cqueue* or *equeue*. If a packet reaches the last ihandler, it will be either delivered to the next hop or to the user application.

Users can insert their own ihandlers into SRouter to process incoming packets. SRouter has a very simple but powerful mechanism to load user-defined ihandlers. User only needs to specify the path of the folder containing all the ihandlers in job description archive. After LiteLab starts a job, SRouter will load them one by one, in the specified order.

VID

To be neutral to any naming scheme, LiteLab uses logical ID (VID) to identify a SRouter. A VID can be an integer, a float number or an arbitrary string. By using VID, we do not need to allocate separate IP address for each SRouter. Every VID is mapped to <IP:PORT> tuple and an SRouter maintains a table of such mappings. These mappings are also a key to enabling dynamic migration, because LiteLab can migrate an SRouter onto another node by simply updating the VID mapping table.

Routing

Routing in LiteLab is also based on VID. In LiteLab, an experimenter can use his own routing algorithm either by plugging in ihandlers or by defining a static routing policy. LiteLab has three default routing mechanisms:

1. **OTF**: Uses OSPF [87] protocol. Given the topology file as input, the routing table is calculated on the fly when an experiment starts. The routing table construction is fast, but the routes are not symmetric.

2. **SYM**: Symmetric route is needed in some experiments. In such cases, LiteLab uses *Floyd-Warshall algorithm* [88] to construct routing table. In the worst case, *Floyd-Warshall* has time complexity of $O(|V|^3)$

and space complexity of $\Theta(|V|^2)$. Therefore, the construction time might be long if the topology is extremely complex.

3. **STC**: This method loads the routing table from a file, avoiding the computational overheads in the other two methods and giving full control over routing.

User Application

ihandler provides a passive way to interact with SRouters. Besides ihandler, SRouter has another mechanism for users to interact with it: *user application*. This feature makes it possible to use SRouter as a functional end-system. In the beginning of an experiment, LiteLab will also start the user applications running on SRouters after they successfully load all the ihandlers.

SRouter exposes two interfaces to user application: *send* and *recv*. Equivalent to standard socket calls, a user application can use them sending and receiving packets. Currently, we have a synchronous version implemented, an asynchronous version is in our future work. User applications can also access various other information like VID, routing table, link usage, etc.

In a nutshell, LiteLab is highly customizable and extensible. It is very simple to plug in user-defined modules without modifying the code and substitute default modules. SRouter can also be used as end-system instead of simply doing routing task. When used as end-system, user-implemented applications can be run on top of it.

Features & Limitations

We performed thorough evaluation on LiteLab to test its accuracy, performance and flexibility. In following sections, we present how we evaluates various aspects of LiteLab and how we adapted it to get around practical issues. We also give some use cases to illustrate the power of the platform.

Accuracy: Link Property

In terms of accuracy, we have two concerns with using software-simulated router on general purpose operating system. First, is SRouter able to saturate the emulated link if it is operating at full speed? Second, can SRouter emulate the link properties (delay, loss rate and bandwidth) accurately?

We performed a series of experiments to test the accuracy and precision of SRouter. We used different values for bandwidth, delay, packet loss, and

packet size in the evaluation. Our test nodes run Ubuntu SMP with kernel 3.0.0-15-server. The operating system clock interrupt frequency is 250HZ. We set up an experiment where we use two SRouters as end-systems, each running both a server and a client.

In bandwidth limit experiments, we used one-way traffic. A server keeps sending packets to a client on the other node. Table 3.1 summarizes the experiment results. With 1518-byte packets, SRouter can easily saturate a 100 Mbps link. With 64-byte packets, two directly connected SRouters can exchange 10000 packets (625 KB) per second. This low number stems mainly from our use of Python to implement LiteLab. Although C would be faster, we opted for Python in the interest of simplicity and flexibility. We also tested a multi-hop scenario and observed only negligible additional decreases in bandwidth.

Compared with the results in [86], SRouter is much better than *nse* [89] and close to dummynet. One reason why dummynet has slightly better accuracy is it increases the clock interrupt frequency of the emulation node to 10000HZ, 40 times of ours, which improves the precision accordingly. Another reason is that dummynet works at the kernel level thus has no user-level overheads. Based on the results, SRouter makes a reasonable tradeoff between accuracy and complexity. It shows that application layer isolation is able to provide satisfying accuracy and precision. We can increase experiment scale greatly without sacrificing too much reality.

Table 3.2 summarizes the results from delay test. We used the same topology as in the bandwidth limit test, with the difference that traffic is two-way and there is no bandwidth limit. In an ideal situation, the observed value should be twice the set value. The results show that as the delay increases, the error drops even though the standard deviation (stdev) also increases. Both small and large packets suffer from large error rate when the delay is less than 10ms. We also noticed that a high-speed network (10 Gbps) can provide better precision than a low-speed network (1 Gbps) in both experiments and comparing with previous work [86].

Table 3.3 summarizes the experiments for packet loss observed by the customer. The accuracy of the modeled loss rate mainly relies on the pseudo-random generator in the language.

Scalability: Topology

Being able to quickly construct new topologies certainly improves efficiency. Compared to Emulab and PlanetLab, LiteLab's configuration and setup is lighter. Nodes are identified with VIDs and no additional IP addresses are needed.

Table 3.1: Accuracy of SRouter's bandwidth control as a function of link bandwidth and packet size.

BW (Kbps)	Packet Size (bytes)	Observed Value	
		BW (Kbps)	% err
56	64	55.77	0.41
	1518	57.62	2.89
384	64	382.56	0.37
	1518	387.96	1.03
1544	64	1539.23	0.31
	1518	1546.32	0.15
10000	1518	9988	0.12
45000	1518	44947	0.12

Table 3.2: Accuracy of SRouter's delay at maximum packet rate as a function of 1-way link delay and packet size.

OW Delay (ms)	Packet Size (bytes)	Observed Value		
		RTT	stdev	% err
0	64	0.190	0.004	N/A
	1518	0.221	0.007	N/A
5	64	10.200	0.035	2.00
	1518	10.230	0.009	2.30
10	64	20.212	0.057	1.06
	1518	20.185	0.015	0.92
50	64	100.209	0.060	0.21
	1518	100.218	0.031	0.22
300	64	600.189	0.083	0.03
	1518	600.273	0.034	0.04

To test how fast LiteLab can construct an experiment network, we used both synthetic and realistic topologies, deployed on 10 machines. For realistic topologies, we used 4 ISPs' router-level networks from Rocketfuel Project [84]. Table 3.4 shows the time used in constructing these networks using two different routing table computing methods (OTF and STC from Section 3.5). The result shows LiteLab is very fast in constructing realistic networks, most of them finished within 5 seconds. Even for the largest network NTT, the time to construct is only about 10 seconds.

In some cases, the experimenters need symmetric routes. As we mentioned in Section 5.2, network constructed with OTF cannot guarantee

Table 3.3: Accuracy of SRouter's packet loss rate as a function of link loss rate and packet size.

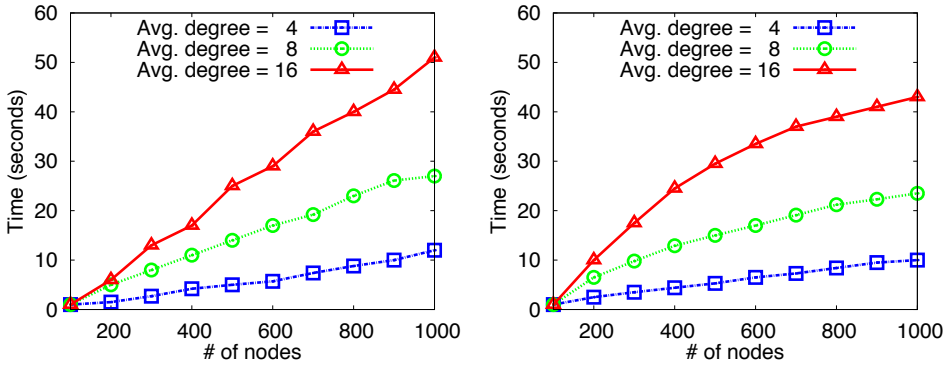
Loss Rate (%)	Packet Size	Observed Value	
		loss rate (%)	% err
0.8	64	0.802	0.2
	1518	0.798	0.2
2.5	64	2.51	0.4
	1518	2.52	0.8
12	64	11.98	0.1
	1518	11.97	0.2

Table 3.4: Time to construct realistic ISP networks. OTF: routing table is calculated on the fly; STC: routing table is pre-computed and loaded by routers

ISP	# of routers	# of links	OTF	STC
Exodus	248	483	1.5s	16s
Sprint	604	2279	4.6s	141s
AT&T	671	2118	4.2s	204s
NTT	972	2839	10.1s	312s

symmetric routes. SYM is impractical for complex topologies, so STC is the only option, although much slower than OTF. The first bottleneck is transmitting the pre-computed routing file to the local machine; the second bottleneck is loading the routing table into the memory. There are several ways to speed up STC: first, storing the routing file in local file system; second, using more physical nodes.

We also used synthetic network topologies in the evaluation. The purpose is to illustrate the relation between construction time and network complexity. We chose Erdős-Rényi model to generate random network and Barabási-Albert model to generate scale-free network. Figure 3.6 shows the results of 50 different synthetic networks with the different average node degrees. The number of nodes increases from 100 to 1000. In the biggest network, there are about 16000 links. From the results, we can see given the node degree, the time to construct network increases linearly as the number of nodes increases in random network. However, the growth of time is slower in scale-free network because the nodes with high degree dominate the construction time.



(a) Random (Erdős-Rényi) (b) Scale-free (Barabási-Albert)

Figure 3.6: Time to construct synthetic networks of different type.

# of PHY	# of SR	Naive (s)	Heur (s)
128	100	1.30	0.03
128	200	3.23	0.08
128	400	5.37	0.23
128	800	11.61	1.00
256	100	1.93	0.03
256	200	4.62	0.08
256	400	9.47	0.23
256	800	22.24	0.98

Table 3.5: Efficiency of LP solver as a function of different network size. **PHY**:physical nodes, **SR**:SRouters

Adaptability: Resources Allocation

As mentioned, we use an LP solver to map virtual nodes to physical nodes. The LP solver module uses *CBC*[2] as engine, takes node states and job requirements as inputs, and outputs a deployment matrix. We tested how well our LP solver scales by giving it different size of inputs. Table 3.5 shows the results.

The size of deployment matrix is the product of the number of physical nodes and the number of SRouters. The results (*Naive* column in Table 3.5) show that as the deployment matrix grows, the solving time increases. It implies that even for a moderate overlay network, solving times can be significant if there are a lot of physical resources.

[2]https://projects.coin-or.org/Cbc

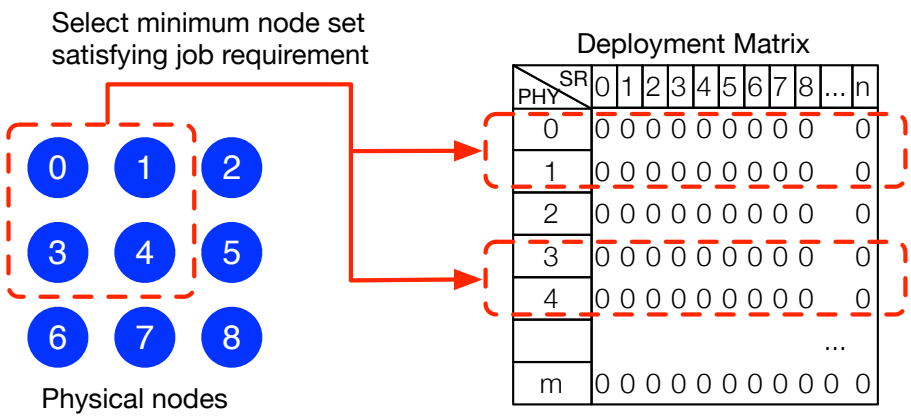


Figure 3.7: Reduce deployment matrix size by selecting minimum physical node set that satisfies the job requirement.

To reduce solving time, we must reduce matrix size. We cannot reduce the number of SRouters in the experiment, but can limit the number of physical nodes. Algorithm 1 shows our heuristic algorithm which attempts to limit the number of physical nodes. The algorithm picks the minimum number of nodes needed to satisfy the *aggregate resource requirements* of the experiment and then attempts to solve the LP. Because the actual requirements like CPU or memory of a single SRouter cannot be split onto two physical nodes, it is possible that the LP has no solution. We then double the aggregate resources required, add more physical nodes, and attempt to solve the LP again. Eventually a solution will be found or the problem will be deemed infeasible, i.e., although the aggregate resources are sufficient, it is not possible to find a mapping which satisfies individual node and SRouter constraints. In our tests, we discovered that the optimal solution is in most cases found on the first try. Figure 3.7 shows how the matrix size is reduced.

The efficiency of the LP solver with heuristic algorithm is also shown in Table 3.5 (Heur column). By comparing with naive LP solver, the solving time is significantly reduced and *it is independent of the number of available physical nodes.*

Application: Use Cases

We have used LiteLab as an experiment platform in many projects. Compared to other platforms, LiteLab speeds up experiment life-cycle without sacrificing accuracy, especially for very complex experiment networks. We

```
1: Input: job requirements R, physical nodes L
2: Output: minimum physical node set S
3: Calculate overall job requirement R
4: Order nodes from lightest to heaviest load into L
5: for all node N in L do
6:    Add N to S
7:    Calculate capacity of S: C
8:    if R < C then
9:       Solve LP
10:      if optimal solution exists then
11:         break
12:      else
13:         R ← 2 × R
14:      end if
15:   end if
16: end for
```

Algorithm 1: Heuristic to improve mapping efficiency

have tested LiteLab in the following situations:

1. Router experiment: new queueing and caching algorithms can be plugged into LiteLab and test its performance with various link properties.

2. Distributed algorithms: LiteLab is also a good platform for studying distributed algorithms, gossip and various DHT protocols can be implemented as user applications.

3. Information-centric network experiments: various routing and caching algorithms can be easily tested on complex networks, using realistic ISP networks.

Limitations

LiteLab aims at being a flexible, easy-to-deploy experiment platform and in this goal, it must make some tradeoffs regarding accuracy and performance. In terms of accuracy, the main factor is the precision of the system interrupt timer, especially when simulating low-level link properties. Increasing timer frequency, like in [86], will improve accuracy, but requires root privileges and possible recompilation of the kernel. LiteLab runs in user space and does not require root privileges.

Overall system load will also affect LiteLab's performance. LiteLab attempts to avoid these issues by selecting lightly loaded nodes and migrating tasks from heavily loaded nodes, however it cannot completely eliminate external effects from other processes running on the test platform. Any platform on a shared infrastructure suffers from this same problem and the only solution would be to use a dedicated infrastructure.

SRouter's processing power is another limitation as it can only process about 10000 packets per second. Adding more user-defined modules will further slow down SRouter. This limitation stems mainly from our choice of Python as implementation language and a C implementation would yield better performance. Using Python has made the development of LiteLab a lot easier and less error-prone than using C. This means that LiteLab is not well-suited for low-level, fine-grained protocol work. However, for studying system-level behavior and performance of a large-scale system, Litelab is better suited than existing platforms.

Other Tools

We now compare features and capabilities of LiteLab with the three other existing approaches. LiteLab is a time- and space-shared experiment platform. It leverages the existing nodes available to the experimenters and attempts to maximize utilization of all available physical resources.

Compared to NS2/3 (and other simulators like [90–92]), LiteLab runs over a real network and allows deployment of user applications on top of the experiment platform. LiteLab allows experimenting with very large topologies with relatively little physical resources. Specific-purpose simulators, e.g., [93–96], are lighter than NS2/3, but are limited to a single application. Parallel simulation [97] may offer a solution to the scalability issues of simulators.

Compared with Emulab, LiteLab runs on generic hardware and does not require any particular operating system or root privileges. Emulab is more accurate in simulating certain network-level parameters, but LiteLab is able to run a much larger experiment with the same hardware because multiple SRouters can run on a single physical node. Work of Rao et al. [98] is close to our approach. However, their work focuses on a specific application whereas LiteLab is a generic network experimentation testbed.

LiteLab is very similar to PlanetLab, with a few key differences. PlanetLab runs on a dedicated infrastructure whereas LiteLab can leverage any existing infrastructure. PlanetLab has the advantage of using a real network between the nodes, but at the same time, is not able to guarantee network performance between nodes. LiteLab, on the other hand, can con-

figure the network properties with very good accuracy and allow better repeatability for experiments.

3.6 Conclusion

In this chapter, we have argued that measuring the performance of caching in information-centric networks is fundamentally different from previous networks of caches, like web caching hierarchies. We have discussed different metrics and showed common mistakes in use of metrics in caching in information-centric networks. In addition, we have highlighted some lesser-known metrics which we consider more appropriate, and have proposed new metrics that capture more fine-grained information about the performance of an ICN caching network. Finally, we have considered how ICN caching experiments should be set up and discussed key elements of experimental work. The discussion over the measurement metrics is also complemented with a full description of the experiment platform we used in our evaluation. To summarize,

- We need to evaluate the performance of the network as a whole as opposed to optimizing individual caches.
- Byte hit rate measures only reduction in inter-ISP traffic and due to content providers installing their servers close to clients may not reflect a true reduction in outgoing (costly) traffic.
- Using average hops as a metric is not a good idea since it does not discriminate well and has issues related to assigning the number of hops for cache misses.
- Footprint reduction measures reduction in intra-ISP traffic and gives more information about where the hits happen. It is to be preferred over byte hit rate and average hops.
- A more refined metric, like the coupling factor, can help characterize the performance by looking at several aspects (content popularity and topology in this case) and provide insight into the inherent behavior of a network of caches.
- Experiment setup needs careful thinking and its execution needs tight control in order to get meaningful results.

Chapter 4

Neighborhood Search and Admission Control

In-network caching of content is a popular technique for eliminating redundant traffic from the network and improve the performance of network applications. In this chapter we present a novel cooperative caching strategy to improve performance of in-network caches. Our cooperative scheme is composed of an admission policy for the incoming data and a content exchange protocol between neighbor network caches to improve the search zone. The admission policy enforces that a previously cached data is not unnecessary replicated in other caches, resulting in more space for new data. The content exchange protocol allows for exchange on cached data, increasing the hit rate for incoming requests. The benefits are twofold: first, we reduce the redundant content caching in the network, and second, we improve the hit rate by informing the content cached in the nearby caches. As a proof-of-concept, we have implemented a prototype and evaluated its performance using different large-scale topologies against standard non-cooperative caching algorithms. Our numerical results show that both admission and content exchange policies yield large performance gains over standard algorithms.

4.1 Introduction

The hype with user generated content (UGC) such as Youtube videos and IPTV, has put the current Internet infrastructure under high burden. According to a Cisco survey [99], the global IP traffic is expected to grow four times from 2009 to 2014, approaching 64 Exabytes per month in 2014, and by that time, various forms of video (TV, video-on-demand, and P2P) will

exceed 91% of the traffic.

Despite these predictions, ISPs lack incentives to upgrade their infrastructure, an issue known as the *middle mile* problem [100]. The middle mile is the infrastructure that interconnects the transit points between different ISPs, and ISPs do not have enough incentives to upgrade this infrastructure because they do not receive any direct revenue from that upgrade. On the flip side, ISPs have incentives to upgrade the *last mile*, which is the infrastructure that connects subscribers to the Internet. Also, content providers have incentives to upgrade the *first mile* to provide better service availability and user experience.

In order to reduce the immediate upgrade pressure in the infrastructure, ISPs have deployed Web caches [38] in their networks to reduce the redundant traffic passing through their networks. Caches provide a simple but effective storage mechanism to reduce the latency and bandwidth usage. Content delivery networks (CDNs) have been deployed in the Internet to improve the user experience by placing content in the client side of the network, i.e., in the same ISP or close to it, reducing the latency and eventual bandwidth bottlenecks between ISPs. There are two main incentives for the usage of caches in the Internet. First, storage prices have decreased substantially faster than bandwidth costs. Second, data consumption is time-correlated in the Internet, i.e., a given piece of data is produced once and consumed many times in the Internet following a Zipf probabilistic distribution [31]. Recent surveys [101, 102] confirm that a large portion of the network traffic is redundant and could easily be cached.

In this chapter, we propose a cooperation protocol for network caches to improve the caching capacities. The general idea is to use content routers to provide routing and caching capabilities together in the network. Additionally, these content routers implement a novel admission control in the caches, described as *cached-bit*, and a content look up procedure described as neighbor search. The cached-bit strategy reduces the redundant data in the network through a bit set in the content header and the neighbor search procedure allows for content discovery in the neighbor caches, resulting in footprint reduction. As a proof-of-concept, we implemented the caching network together with these caching strategies and evaluated them experimentally through emulation using Rocketfuel topologies. We compare the bandwidth savings, cache hit and the control overhead against simpler models.

The organization of this chapter is as follows. Section 4.2 describes the background on in-network caching and Bloom filters, then proposes the routing mechanism based on exchanged cache digests. Section 4.3 presents

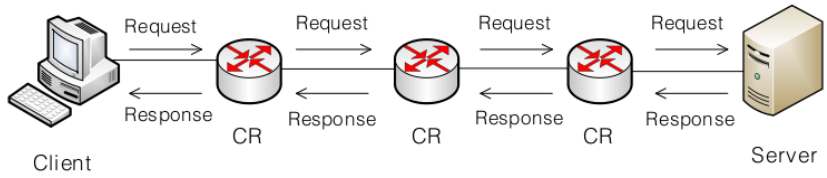


Figure 4.1: Basic store-n-forward content router.

the prototype implementation and describes the evaluation in several scenarios. We present and discuss the results in Section 4.3. Finally, Section 4.5 summarizes the chapter.

4.2 Heuristic Cooperative Caching

Our cooperative caching strategy is composed of an admission policy and a content discovery protocol. In the following, we start with the background of cooperative caching. Then we describe the admission policy, followed by the neighbor search scheme with Bloom filters and the combination of both of them.

Background

The basic in-network caching mechanism is performed by a *content router* (CR) [103]. A CR is a data forwarder similar to a regular router, but it also has internal memory that can be used to store data in transit. The simplest model works as follows: first, for each data response, any CR on the path between a server and clients caches the data in their memory. Further requests can be served by the local copy in the CR. This model is simple and works just as a network storage mechanism with a simple income queue, presenting some benefits such as reduction in the content retrieval latency and bandwidth usage. Fig. 4.1 illustrates an example of CR in a network topology.

The simple CR model has limitations, especially in scenarios where they need to work as a single system. First, the usual admission policy is basically to *cache everything that is possible*, meaning that all in-transit data should be cached in the CR. However, CRs have limited storage capacity and they need more fine grained admission control policies to filter the insertion of new entries in the caches. Second, there is no cooperation between CRs, leading them to cache the same piece of content in the network. For example, a line of CRs between a client and a server will cache the same

content, reducing the effectiveness of the caching mechanism. Therefore, we focus on two main topics: cache admission policies and cache cooperation strategies.

In [103], we proposed the general idea of a neighbor search mechanism that improves the cache hit ratio in cooperative caching networks. The idea is to allow queries to be diverted to neighbor caches, resulting in higher hit rate. However, this strategy alone does not improve the hit rate much due to the fact that all caches tends to have the same content in a linear path. Therefore, neighbor searching in CRs that have the same content results in poor performance. Another limitation of the old neighbor search strategy is that each CR holds a pointer to the CR that has the content, requiring additional memory to store neighbor information. In this chapter we propose a new admission policy that improves the efficiency of storage space use and propose a new neighbor search algorithm combined with Bloom filters to store neighbor information. Bloom filters are space efficient structures that allows for aggregated neighbor information storage, resulting in higher hit ratio. Bloom filters are detailed below.

Admission Policy

An admission policy of a CR decides whether a piece of data should be cached or not. For network caching scenarios, we need a simple yet effective way to decide the admission since the CRs work as routers and need fast decisions. According to [31], the cache hit rate grows logarithmically with the cache size. Thus, it is interesting if we could aggregate the cache sizes by removing all the redundant content in these caches.

As a solution, we propose the *cached bit* admission control. The cached bit is a single bit set in the header informing whether a given piece of data has already been cached in the network or not. Whenever there is a message carrying some data, the first CR that caches the data also sets the bit in the header, informing further CRs along the path that piece of content has already been cached in the network. Therefore, other CRs know that they do not need to cache that piece of content again. The benefit of this approach is that the overall caching capacity can be improved in the network. Compared to the *cache all* admission policy, the cached bit can reduce the amount of redundant data in the network. The cached bit policy solves the previous limitation of the CR model in [103], where caches along the path had the same content. In this approach, just one CR has a given piece of content. Fig. 4.2 illustrates the basic CR model with the cached bit strategy.

In this example, the content server answers with data to client one (red

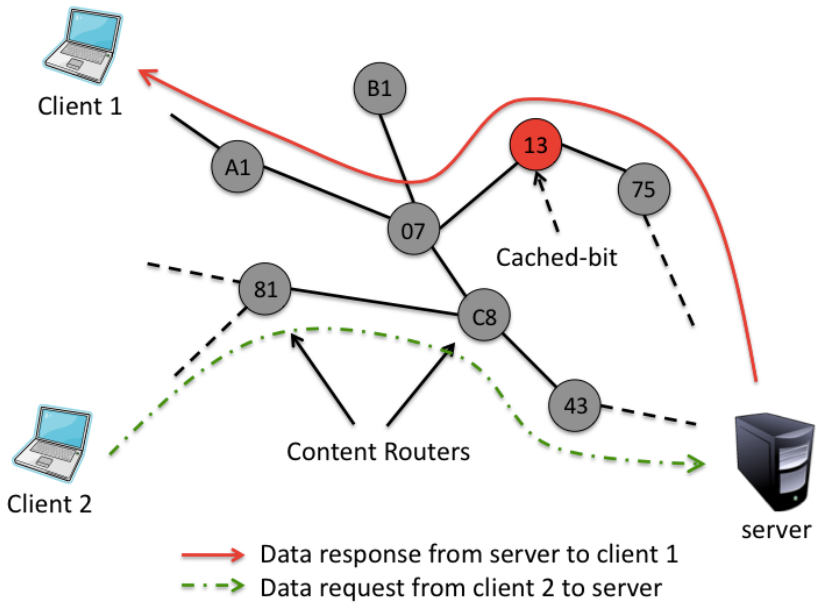


Figure 4.2: Basic cache-and-forward with Cached Bit strategy.

line). The CR with $id = 13$ caches the data, thus, neither CR with $id = 07$ nor CR with $id = A1$ will cache the same data again. Thus, the cached bit reduces the amount of replicated data along a network path. Despite the improvement with the cached bit policy, we can see that if there is a second client (client 2) located in another edge of the topology, she will not benefit from the cached bit policy.

Neighbor Search with Bloom Filters

The Neighbor Search (NbS) strategy is a cooperative scheme to improve the hit rate in the cases shown in the above example where subsequent requests for the same content do not follow the path of the original request. This increases chances of finding the content and improves hit rate and reduces network traffic. The basic model is illustrated in Fig. 4.3.

Each CR has a *neighbor table*, where each entry contains a content ID and the interface where the data came from. Hence, upon receiving a content request, the CR looks up in the neighbor table for the entry, and if positive, it forwards to the interface where the data was last seen. We use *Content Bloom Filters* (CBF) to store content identifiers from neighboring CRs. Each CR broadcasts to its immediate neighbors a Bloom filter

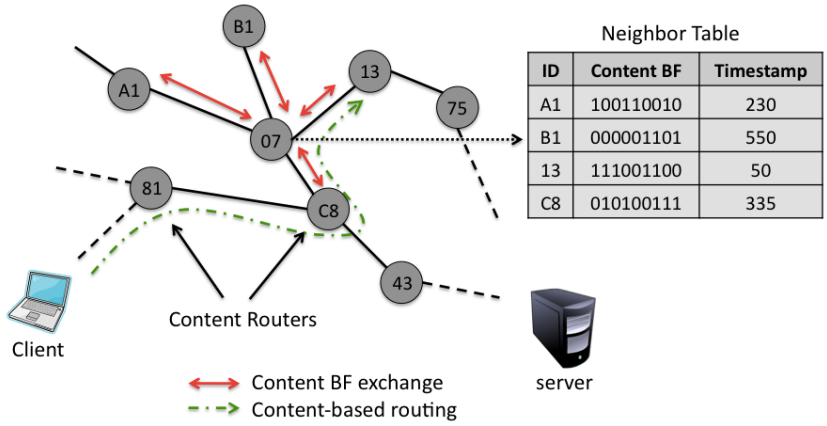


Figure 4.3: Content BF distribution in a network.

informing its currently cached content. Receiving neighbors add this CBF in their neighbor tables, allowing for opportunistic content routing towards CRs that have the content. The benefit of the augmented neighbor search model with Bloom filters is to allow re-directions towards CRs that may have the content with much high probability. Effectively this aggregates the storage of all neighbor CRs, and a request does not query a single CR, but a set of CRs in the network.

Fig. 4.4 illustrates how CBFs can be used to take the routing decisions. First, the content ID is hashed using the internal hash functions (three in this example). Then it will generate a BF mask that is going to be used against the neighbor CBFs stored in memory. In this example, the CRs stores the neighbor CBFs and neighbors-of-neighbors CBFs, indicated by the distance in the number of hops.[1]

CBFs have some drawbacks, *false misses* and *false hits*. False miss is when a piece of content is present in a CR, but the CBF does not reflect it. A false hit is when a CR does not contain the data anymore, but the CBF still has it.

Neighbor Search with Cached Bit

Combining the cached bit admission policy with the neighbor search cooperation strategy aims to improve the cache hit rate further. Cached bit reduces redundant data in the network. As the hit rate is logarithmically proportional to the cache size, the reduction of the replicated data increases

[1]The number of hops can be tuned depending on the amount of memory available in the CRs.

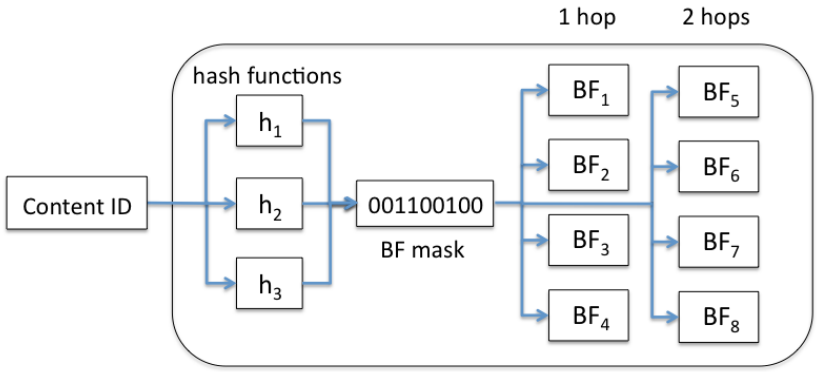


Figure 4.4: CR neighbor look up table.

the storage capacity and the hit rate as well. Neighbor Search increases the search space by aggregating a set of CRs in one larger storage. As consequence, a single query may implicitly go through several CRs for a cache hit. CRs work both cooperatively to reduce the redundant data and also increase the query space.

Refreshal Procedure

The CBF generation and refreshal is triggered by two events. First, at regular intervals, CRs generate CBFs and advertise them, for example, every second. The main benefit of this approach is that the overhead generated by the CBF exchange is known *a priori* and can be used in a network management system. CRs can also be configured to invalidate CBFs after that interval, thus, reducing problems with failovers. The drawback of this approach is that CRs do not analyze the data in-transit, for example, in-transit content refreshal. In this scenario, if the data changes too fast, the source CR is not able to update the CBF quickly to reflect its current state, increasing the number of false misses. Conversely, if changes are slow, there will be a number of unnecessary updates with small changes in the CBF.

Second, a CR sends an updated CBF when a certain number of events have happened such that, e.g., a given fraction of content in the CR is not represented in the CBF. The benefit of this approach is that it adapts to the real network conditions, for example, if suddenly the network changes due to popular content, it can quickly update its neighbors. However, this means that the amount of network traffic is unpredictable, since a flash crowd will trigger a lot of CBF updates. Fig. 4.3 illustrates an example of CBF broadcast to the immediate neighbors. The CRs populate their

routing tables with the CBFs which are used in the routing decision towards the most likely location where the content is located.

Miscellaneous

The bootstrapping procedure is straightforward: CRs periodically broadcast CBF to its immediate neighbors, informing which content it has. Receiving CRs add an entry in the neighbor list and use it for further forwarding decisions. Failovers can also be detected in the same way. Each CBF has a validity associated and when it expires, the CR invalidates that entry and does not use that entry for the routing purposes.

Extending the query area represented by the CBFs would be possible when allowing CRs to forward CBFs from other CRs. This may result in routing loops so we would need to add identifiers in CBFs to indicate which is the original source CR so that CRs receiving duplicate CBFs would be able to drop them.

4.3 Implementation & Evaluation

We have implemented a CR prototype in Python. Our evaluations are performed on self-developed experiment platform, the core of which is software-implemented router. Software-implemented router simulates a realistic router. On our experiment platform, each router and the links between them can be configured individually in terms of link bandwidth, delay, loss rate and so on. User modules like queuing policy and caching strategy can be easily plugged into router.

Fig. 4.5 shows the packet header used in the neighbor search mechanism. The header has six fields and it was designed to be aligned to 32 bits. The *type* field has 4 bits that defines the type of the message. The *TTL* has a 8-bit field defining the time-to-live of the message. The *loop-BF* is a 116-bit field containing a BF of all IDs of CRs that the message has been through, in order to prevent loops. The *CR_ID* is a 128-bit field containing the CR_ID of the source CR. The *nonce* is a 128-bit field containing a number that is used as message identifier between CRs, i.e., a receiving CR can distinguish the order of the received messages in order to update the neighbor table. The *CBF* is the content BF containing a set of content IDs aggregated in the BF, to be used as neighbor information.

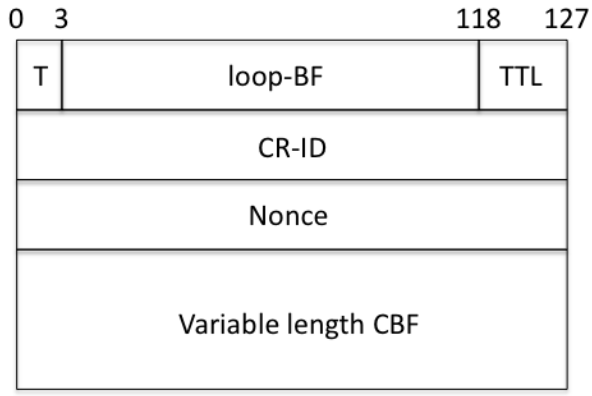


Figure 4.5: Neighbor search header.

Testbed Set-up

All the experiments are performed on our department cluster consisting of 240 Dell PowerEdge M610 nodes. Each node is equipped with 2 quad-core CPUs, 32GB memory, and connected to a 10-Gbit network. All the nodes run Ubuntu SMP with 2.6.32 kernel. The experiment platform will allocate necessary physical resources according to the simulated ISP network size. If ISP's network size is larger than the actual cluster size, routers will be multiplexed onto one node.

Methodology

We use the following metrics to evaluate our proposal:

- **Hit Rate:** What fraction of requests was served by CRs. Because all objects are of the same size, the hit rate is also the byte hit rate.
- **Average Number of Hops:** Average number of hops that a request need to go in the network before finding a cache or a server that holds a copy of the requested data.
- **Footprint Reduction:** Network footprint is the product of the amount of data and the network distance from which the data was retrieved. It measures the amount of internal traffic reduction, i.e., a smaller footprint (larger reduction) means less traffic within the ISP's network.

In the experiments, we used two POP-level topologies from real ISP networks [104], the Sprint and AT&T networks. For the deployment, in each network, we selected the top 20 routers with highest degrees as servers, and the rest as clients. The clients keep requesting data from the servers in the experiment, and the requests are uniformly distributed to different servers.

The request pattern follows Zipf distribution with $\alpha = 0.9$, which is very close to real-life distribution shown in [102]. We also tested two traffic patterns, one is the constant traffic rate, while the other follows a gravity model used in [105]. In gravity model, the fraction of the traffic from each client is determined by the city population. However, no significant difference in results has been found in the evaluation.

Our request trace requests chunks of content, which are assumed to be independent of each other (the popularity of chunks follows the above Zipf distribution). We assigned each CR with storage capacity of a certain number of chunks and assumed that every CR in the network had the same amount of storage. We report our results as a function of the per-CR storage, which at the largest sizes (1024 chunks) was around 1% of the total amount of content.

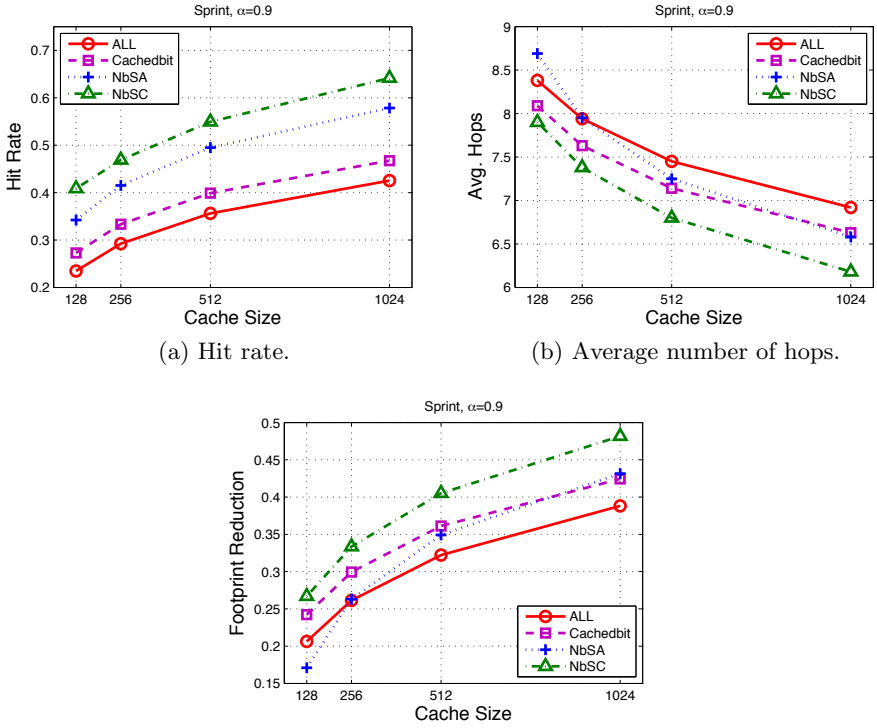
Experimental Results

We used the three metrics explained above: *hit rate, average number of hops* and *footprint reduction*. For the experiments, we use 4 different strategies as defined in Section 4.2:

- **ALL**: no admission policy, CR caches everything
- **Cachedbit**: one CR caches the content along a path
- **NbSA**: Neighbor search with ALL policy
- **NbSC**: Neighbor search with Cachedbit admission policy

Figures 4.6 and 4.7 show the results for Sprint and AT&T networks, respectively. The graphs show hit rate, average hops, and footprint reduction. X-axis is the per-CR storage capacity.

Concerning hit rate, ALL strategy has the worst performance because all CRs along the same path store the same content. Therefore, if there is a miss in one CR, then, it is likely that all CRs in the same path will result in cache miss as well. Cachedbit stores at most one copy per path, so it is better able to use the storage and a miss at one CR might be a



(a) Hit rate.

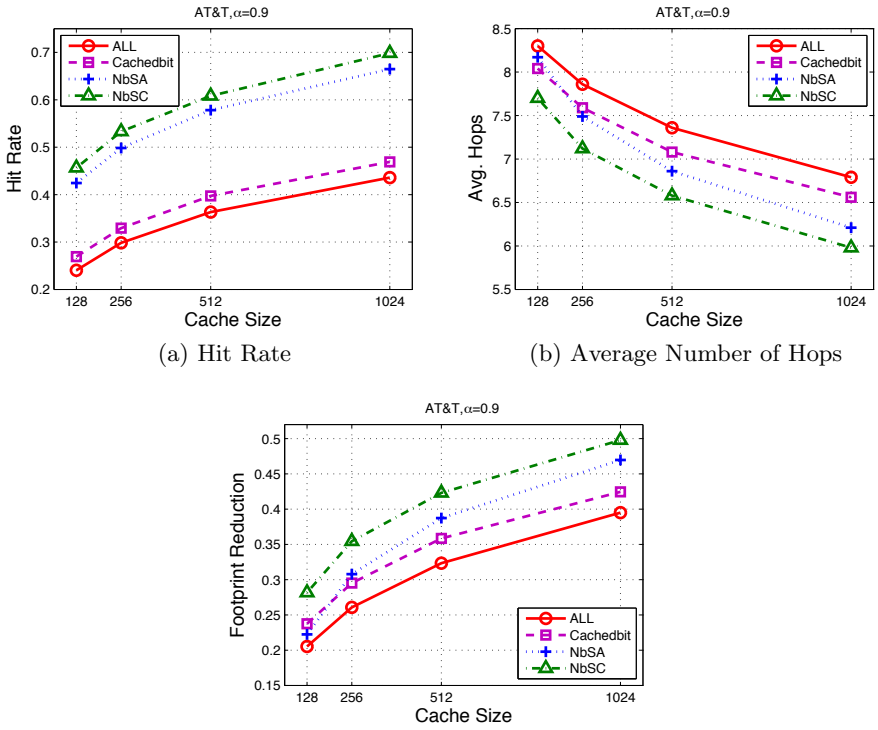
(b) Average number of hops.

(c) Footprint reduction.

Figure 4.6: Comparison between ALL, Cachedbit, NbSA and NbSC in the Sprint Network.

hit in the next CR. NbSA strategy allows for content lookup within a set of CRs, resulting in a combination of a CR with all its neighbors, thus, increasing the search domain. As a result, it has a better performance than ALL and Cachedbit. The best performance comes from the combination of Cachedbit and Neighbor search since Cachedbit reduces redundant data. Compared to ALL, NbSC can increase the hit ratio up to 50% at a small cost.

Concerning average hops, the same ranking between the strategies holds. One point of importance is the behavior of ALL and NbSA for small cache sizes. When encountering a miss, NbSA searches around for the content, but because of the small cache sizes, the other CRs are unlikely to have the content, hence the searching actually costs a lot of traffic. Therefore, *neighbor search strategies might not perform well for small caches.* However, it performs better as the cache size grows. Again, the combination of



(a) Hit Rate

(b) Average Number of Hops

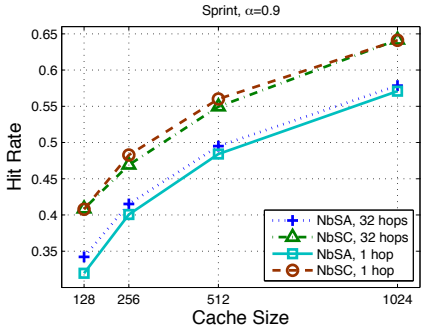
(c) Footprint Reduction

Figure 4.7: Comparison between ALL, Cachedbit, NbSA and NbSC in the AT&T Network.

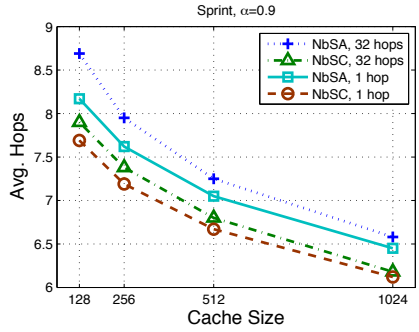
Cachedbit and Neighbor Search results in the smallest number of hops.

As for footprint reduction, we see the same behavior and ranking as with hit rate and average hops, including NbSA's weakness with small cache sizes. Again, the performance of NbSC is the best of them.

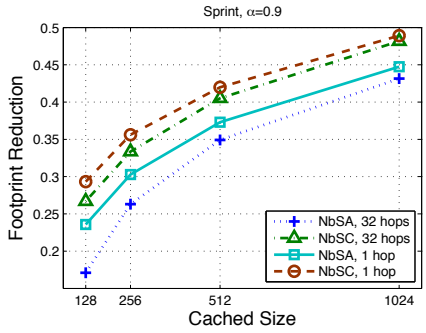
In the next set of experiments, we evaluate the performance of neighbor search versus how many hops away we look for cached content. Results are in Figures 4.8 and 4.9 for Sprint and AT&T, respectively. As expected, querying a larger set of neighbors yields a slightly higher hit rate, and at the cost of increased network traffic. However, hit rate improves much faster by increasing the per-CR storage than extending the search radius, thus there is little benefit from searching for content from far away. The ranking between NbSA and NbSC remains the same with NbSC having the advantage, even with the larger search radius. This underscores the importance of the admission policy.



(a) Hit rate.



(b) Average number of hops.

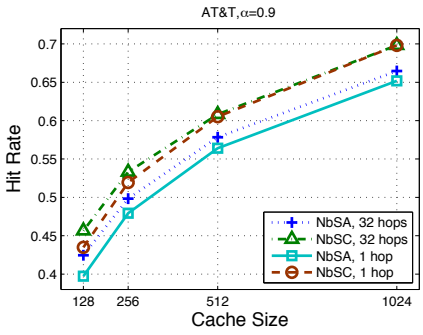


(c) Footprint reduction.

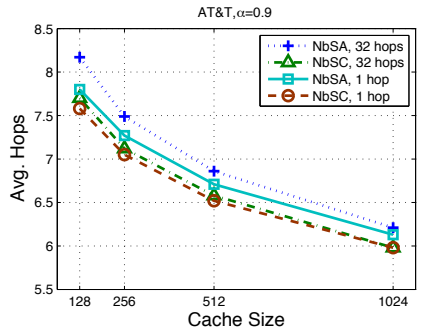
Figure 4.8: Search zone effects for 1 hop and 32 hop search radii and both admission policies, Sprint network

In the next experiment, we analyze the impact of the Bloom filters and the false positive rate. All the previous experiments have been performed with a 1% false positive rate. False positives happen when a CR has a content Bloom filter that is unsynchronized with the source CR. Therefore, the CR forwards the request to a peer CR that actually does not have that content anymore, resulting in cache miss.

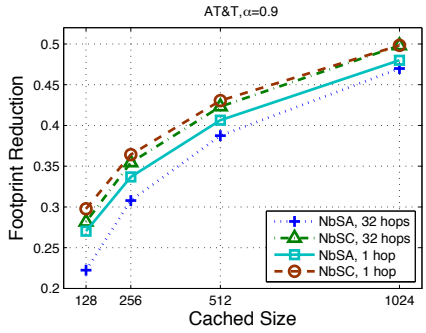
We experimented with 0.1%, 1%, and 5% false positive rates for both neighbor search mechanisms, with 1 hop search radius. We found out that the difference in performance between 0.1% and 1% false positive rates was negligible, i.e., it is not necessary to use very large bloom filters in an attempt to minimize the number of false positives. Going to a 5% false positive rate had a negative effect on performance, in particular on average hops and footprint reduction, but less so with hit rate. For example, for the AT&T network with CR cache size of 512 chunks, the footprint reduction



(a) AT&T Hit Rate



(b) AT&T Average Number of Hops



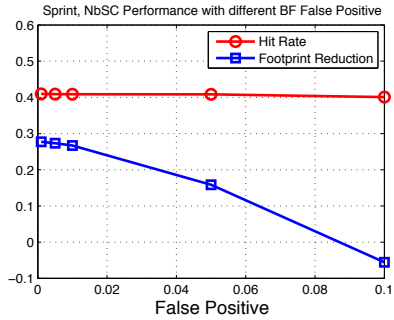
(c) AT&T Footprint Reduction

Figure 4.9: Search zone effects for 1 hop and 32 hop search radii and both admission policies, AT&T network

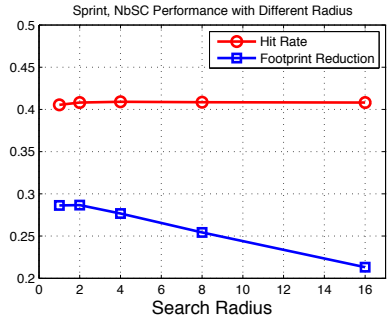
was 40% and 24% for false positive rates of 0.1% and 5%, respectively.

The BF overhead is $m = -\frac{n \ln p}{(\ln 2)^2}$, for a given false positive rate p and number of entries n. Assuming a cache size C and an average file size F, the number of bits needed by a BF is equal to $m = -\frac{C \ln p}{F(\ln 2)^2}$. In terms of memory consumption q, namely the percent of cache needed to store the BF, is $q = \frac{km}{C} = -\frac{k \ln p}{F(\ln 2)^2}$, assuming each router has k neighbors on average. The formula shows that for larger file sizes, we will have smaller BF and vice-versa. Also, additional neighbors require more memory to store the neighbors' BFs.

Figure 4.10 expands on the results on effects of bloom filter false positive rate and search radius on hit rate and footprint reduction for NbSC in the Sprint network. Results for AT&T network and NbSA are similar and are not shown. The results show that the hit rate remains practically the same across different false positive rates but the footprint reduction drops



(a) NbSC Performance vs. BF false positive rates in Sprint network.



(b) NbSC Performance vs. Radius.

Figure 4.10: Performance comparison of NbSC vs. false positive rates and radius, 128 chunks of per-CR storage

as the conditions get worse. This is expected since a cache miss due to an unsynchronized Bloom filter (large false positive rate) will result in additional hops and it does not influence the content cached in the network. As consequence of the increase in the number of hops, the footprint reduction decreases.

Fig. 4.10b shows the comparison of hit ratio, footprint reduction as a function of the search radius. As the results show, the number of searched zones have a small effect when the search zone increases from one to two, but larger search zones result in a drop in footprint reduction, since we search content wider, but often without finding it.

Summary of the Results

We summarize our main findings as follows:

- Admission policy in CRs is very important in ensuring good performance. This is evidenced by the better performance of the strategies using the Cached Bit admission policy vs. the admit-all policy.
- Searching content in neighbor CRs is beneficial, but even a single hop search radius is typically sufficient for getting good gains and increasing search radius yields diminishing returns.
- Bloom filter size (and the associated false positive rate) is not critical and a false positive rate of 1% appears sufficient.

4.4 Comparison to Redundancy-Elimination

Redundancy elimination (RE) techniques have been proposed to handle the huge amount of data in the access networks. Their main aim is to remove requests and/or responses of redundant data in the network, reducing the traffic and costs in the access network.

RE techniques can be classified into two kinds: (a) caching to remove transfers, and (b) data replacement with a shim header. Former relies on caching network-level objects and storing them temporarily in the network. Caching techniques rely on redundancy of the traffic [106, 107], implying that a large portion of the network traffic is duplicated and could be cached for later requests. Another incentive is that storage prices have decreased faster than bandwidth costs [27].

The second approach replaces redundant data with a shim header in an upstream middlebox (usually close to the server) and reconstructing it in a downstream middlebox before delivering it to the client. Commercial products provide WAN optimization mechanisms through RE in enterprise networks [108–110]. Recently, RE has received considerable attention from the research community [107, 111–113]. In [112], the authors propose a network-wide approach for redundancy elimination through deployment of routers that are able to remove redundant data in ingress routers and reconstruct it in egress routers. However, they also require tight synchronization between ingress and egress routers in order to correctly reconstruct the packet and they also require a centralized entity to compute the redundancy profiles. In [113], the authors propose to use caches in the local host and use prediction mechanisms to inform servers that they have already the following redundant data. However, they are not able to share the cached data among other nodes due to the local characteristic of the cache.

Although both caching and RE have been around in the research community, there has not been any thorough comparison in the effectiveness of the two above-mentioned strategies: in-network caching vs. redundancy elimination. Work in [114] combines in-network caching and RE, but limiting the applicability of the solution to a single content source only.

In the following, we perform a comparison between an in-network caching architecture (INCA) and state-of-the-art RE solutions. Although INCA models a generic network caching architecture, it is effectively CCN-like [1, 115]. However, as we want to understand the performance differences between caching and RE, we do consider low-level protocol details.

Overview of RE

Modern RE schemes use a fingerprint-based data stripping model. Nodes generate a set of fingerprints for each packet in transit, where each fingerprint can be generated over a pre-defined block size. Upon detecting a cached fingerprint, the upstream node replaces the data by a fingerprint and the downstream node replaces the fingerprint with the original data, reducing the overall data transmission over the network. As described in [116], both upstream and downstream nodes need to be strongly synchronized in order to work correctly. A similar approach is presented in [117].

Work in [111, 112] proposes to extend the RE technique to the whole network, i.e., to make RE as a basic primitive for Internet.The main idea is to collect redundancy profiles from the network and use a centralized entity to compute paths between destinations within an ISP with higher RE capabilities. Therefore, data going through these networks have higher RE footprint reduction than going to other paths in the network. Despite the improved RE capacity, it still requires strong synchronization between the upstream and downstream routers in order to work properly.

A third RE approach [113] was recently proposed to overcome the synchronization issue in order to be deployed in data-center networks. As cloud elasticity favors the migration and distribution of work among a set of nodes, it is hard to set up the synchronization between two fixed nodes. Therefore, the main idea of [113] is to create a local cache together with a predictive mechanism to acknowledge already cached data to the server. In this scenario, the service sends a predictive acknowledgement to the server informing that the requested data is already present in the client, thus, removing the redundant data. Despite the improvement over the fixed node requirement, the use of local storage prevents the sharing among other nodes, increasing the overall sharing capacity and hit ratio. Therefore, the RE is not network wide, but for redundant data that may be requested again in the local node.

INCA vs. SmartRE

SmartRE [111] uses two network elements, the redundancy profiler and the redundancy-aware route computation element. The redundancy profiler collects in-transit data statistics in order to create a profile of the most popular data to be the ones to be cached in the routers. The redundancy aware route computation computes the paths based on the content stored in the network in order to optimize the redundancy elimination of the network by solving a linear programming (LP) problem. The benefit of such cen-

Network	FP Reduction
Exodus	27.55%
Sprint	28.79%
AT&T	31.59%
NTT	30.45%

Table 4.1: SmartRE footprint reductions in different networks under ideal conditions

tralized element is the fact that it knows the complete topology and makes it possible to compute a good result for the RE. A totally decentralized SmartRE model is not possible since there must be an entity controlling the synchronization between these points.

SmartRE reduces the network footprint, because the caches between the ingress and egress store parts of the data and the ingress simply indicates which parts a cache is to substitute in a packet. There is no effect on external traffic. The LP solver knows the redundancy profile of the traffic and calculates a caching manifest which indicates which parts of which packets should be decoded at which caches. There is a very strong link between the total amount of storage in the network and the length of the sampling period which defines how long traffic is observed to compute the redundancy profile. According to [111], sampling periods on the order of a few tens of seconds are to be expected to be reasonable.

We implemented SmartRE on top of our CR testbed. We noticed that SmartRE, or rather the LP defined in [111], is very sensitive to the parameters in the model. Small deviations often lead to large differences in performance, typically for the worse. We were able to determine parameters for what corresponds to the settings in [111] and calculated the footprint reductions for the same traffic as with INCA. These ideal footprint reductions are shown in Table 4.1.

Figure 4.11 shows the internal traffic reduction as measured by the network footprint reduction. The y-axis shows the fraction of internal traffic that was reduced by the caches in the CRs. As with the other metrics, the differences between the three admission policies are small. Again, NbSC is clearly superior to Cachedbit which, in turn, is clearly superior to the ALL policy. Footprint reduction is the reason why we tweaked Cachedbit to create a copy of the chunk at the CR closest to the client. Without the additional copy, ALL-policy is better at footprint reduction than Cachedbit. We observed that this additional copying drops the hit rate by a negligible amount, but raises the footprint reduction considerably.

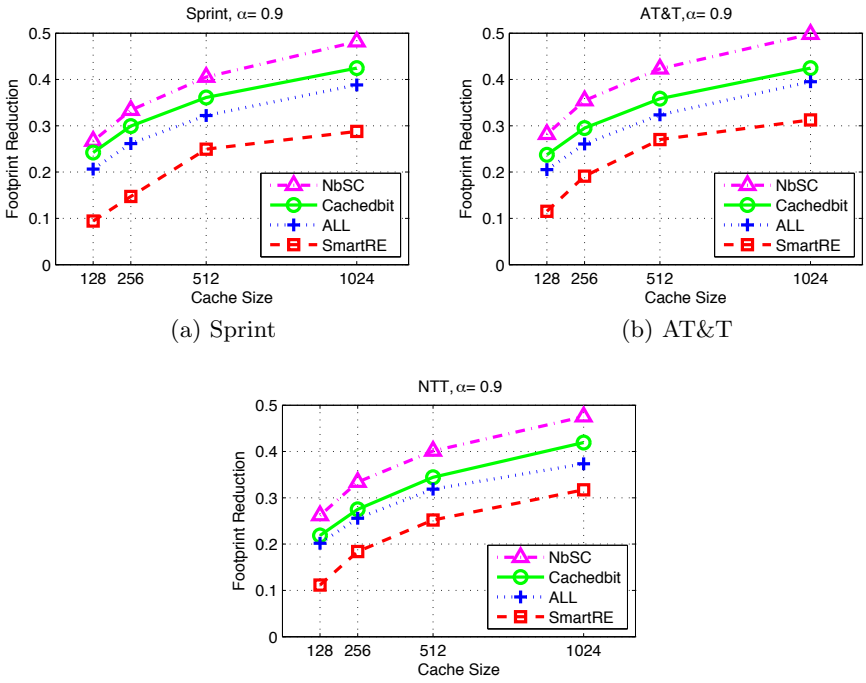
Contrasting the numbers in Table 4.1 with the INCA footprint reductions in Figure 4.11, we see that they are similar in value. For small INCA cache sizes, SmartRE yields a higher reduction, whereas for larger cache sizes, INCA has the upper hand. However, even for very modest cache sizes, NbSC is able to achieve an equal footprint reduction to SmartRE and for large cache sizes, the footprint reduction is improved by 50–65%. Cooperative caching is therefore much more efficient at reducing internal traffic than SmartRE.

Recall that our INCA experiments considered one chunk to represent one file, whereas in the SmartRE experiments, a chunk is one packet. This means that the footprint reduction numbers cannot be directly compared since traffic is different in the two cases. However, based on the numbers presented in [111], we can infer a mapping between SmartRE and INCA experiments. In [111] it is shown that SmartRE gets close to its ideal performance with 6 GB of storage per router. Assuming the same 6 GB of storage per CR, the case of 1024 chunks of storage, where 1 chunk equals 1 file, would imply the average file size to be about 6 MB. If the content is a mixture of text, images, and short videos, this seems like a reasonable, if not even conservative, number. (For content consisting mainly of larger videos, this would not be sufficient.)

We ran experiments with SmartRE where we took the ideal cache size used to obtain the numbers for Table 4.1, and set it to 1/2, 1/4, and 1/8 of that value. For each case, we then ran the experiment to obtain the reduction in footprint. This allows us to plot the INCA and SmartRE footprint reductions on the same x-axis, shown in Figure 4.11. This confirms that INCA is more efficient in reducing internal traffic in the network. The additional reduction in traffic varies between almost 200% for small caches and 50% for large caches.

Cachedbit is similar to the heuristic "Heur1" from [111] in how it attempts to place the content. In [111], the performance of these two heuristics was found lacking when compared to the SmartRE algorithm with its centralized controller deciding on what to cache where. If the same translates to an INCA caching network, a centralized controller deciding on placement of chunks in CRs would be a superior choice. However, similar placement problems are often NP-complete [118], although some simplifications are likely to yield a linear program. We have not considered a central placement agent in INCA, although it could be included in future work.

An important difference is that INCA is able to share cache space between clients, whereas SmartRE has fixed buckets for each ingress-egress flow. This gives INCA more possibilities in exploiting the cached data, thus



(a) Sprint

(b) AT&T

(c) NTT

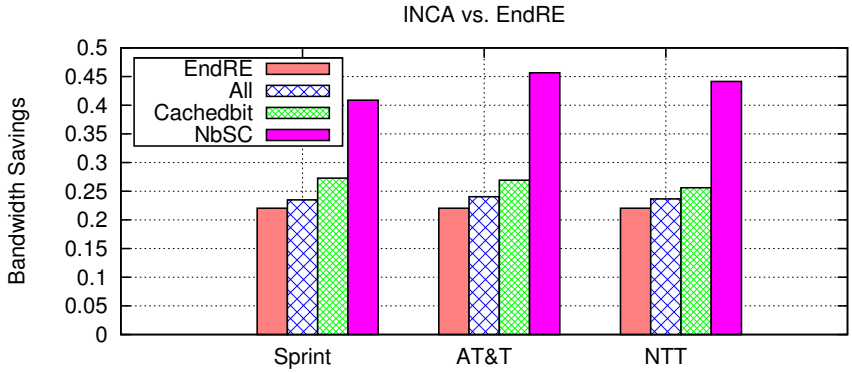
Figure 4.11: Comparing footprints of INCA and SmartRE

reducing footprint and improving hit rate. *We believe this sharing of cache space between all client and server pairs is what gives INCA an advantage over SmartRE.* Contrasting our results to the single server case presented in [114] is part of our future work.

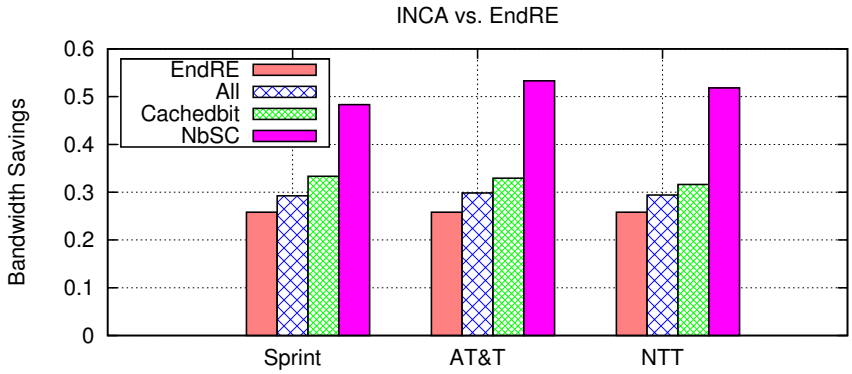
Comparing INCA with SmartRE, we come to the following conclusions:

- For external traffic reduction, INCA is always superior, because SmartRE has no effect on external traffic.
- For internal traffic reduction, performance of INCA (with neighbor search) is in most cases clearly superior, up to 50–65% more reduction in internal traffic. However, the differences depend on how the mapping between cache sizes is done and the file size distribution, thus in different environments the results could be different.

However, in our experimental environment INCA with neighbor search is far more effective in reducing both internal and external traffic.



(a) 128 chunks of storage



(b) 256 chunks of storage

Figure 4.12: INCA vs. EndRE

INCA vs. End-to-End RE

Figure 4.12 shows the bandwidth savings of both INCA and EndRE [117] on three different networks. We show cache sizes of 128 and 256 chunks. The bandwidth savings of EndRE remains the same on three networks because it is end-to-end solution. The network topology does not affect its performance. We can clearly see that INCA is superior to EndRE. Even the ALL strategy is slightly better than EndRE in all three networks. PACK [113] is another end-to-end RE solution, but according to [113], its performance is about 2% worse than EndRE. Larger cache sizes improve INCA's performance; figures not shown due to space limitations. Note that INCA's savings are a combination of results shown in Figures **??** and 4.11.

Anand et al. [107] have evaluated real trace captures and their results

suggest that a middlebox-based solution (i.e., something akin to INCA) has an advantage over end-to-end RE solutions in saving network bandwidth. INCA does have a definite advantage in not requiring synchronization between the server and client and since some content can be served from CRs along the path, we avoid having to do a round-trip to the origin of the content, possibly speeding up the transfer.

Summary of the Results

The key findings in the experiments of comparing INCA and RE can be summarized as follows:

- In terms of reducing external network traffic, INCA is always superior when compared to ISP-internal RE solutions [111]. End-to-end RE solutions [112, 113] can reduce external traffic, but are outside the control of the ISP; furthermore, they are not as effective as INCA.
- In terms of reducing internal network traffic, INCA is in most cases clearly superior to state-of-the-art RE solutions [111], with at least 50–65% improvements in internal traffic reduction.

4.5 Conclusion

In [40,119], the authors present an analytical model for data transfer, bandwidth and caching performance in information-centric networks. The proposed model uses a traffic generator with variable request rate for different parts of the same content, different caching capacities along the same path towards a piece of content and the LRU replacement policy within the network caches. Compared to our model, we use a simpler traffic generator that requests the content chunks based on a Zipf distribution, and our focus is on evaluating the admission policy and the caching strategy, leaving the analysis as future work. As our results show, admission policy is critical for good caching performance.

Internet Cache Protocol (ICP) [120] was proposed to increase the cooperation among Web-caches. Whenever a Web-cache receives a client query, it holds the query and multicasts an ICP message to all peer Web-caches to query for the requested content. Upon receiving the answer from some Web-cache, it stores locally a copy of the requested data and forwards to the data to the client. Although ICP could increase the hit ratio, the protocol increased the bandwidth usage due to the multicast query to all Web-cache peers. SummaryCache [66] was proposed to overcome ICP's limitation by

reducing the number of queries that a Web-cache needed to send. Each Web-cache broadcasts a Bloom filter with its current content to all peers and upon receiving a query, it checks which Bloom filter has the content and sends just to those Web-caches. Compared to our model, CRs can not hold the request while it looks for content since it is in the network level. Hence, the ICP model can not be applied to our model, which is much more constrained in terms of storage and latency.

Content Centric Network (CCNx) [1] is a content-oriented communication model driven by *interests*. Content requests are identified by URLs and they are routed directly based on URLs towards the server. Caches on the path are able to identify these requests and answer with the requested data if they have cached, otherwise, they forwards the requests and mark their interest in the content. Despite the similar approach in the caching model, our proposal implements more efficient caching strategies with cached-bit and neighbor search to increase the caching capacity and efficiency in the network. Our CR model is very close to CCN, but we do not explicitly consider how the requests are sent and interpreted by the CRs.

Content-centric Caching (CONIC) [57] is a clean-slate approach for content retrieval in caches based on Conic Routers (CR). Clients requesting data send requests to the servers and CRs on the path intercept the request and query other clients cache for the content. CONIC works for HTTP data and has look up latency associated for each request.

Cache-and-Forward (CNF) [121] is an in-network caching architecture where routers have a large amount of memory for storage. These routers perform content-aware caching, routing and forward *packet* requests based on location-independent identifiers. In contrast, we provide extra features to improve the cache hit in the network and also content search capabilities. Also, we are not restricted to HTTP as CNF is.

Cachecast [122] proposes a redundant traffic elimination technique in the multicast communication. They place small caches on links that inspects all frames and replaces with a shim header that represents the redundant data and the shim header is translated to the actual data in the destination. Our work is different since instead of removing redundant traffic, we assume that clients can use caches as alternative sources.

These approaches are architectural in nature, and do not investigate the effectiveness of the distributed caching from caching effectiveness point of view, which is our focus.

To summarize, in this chapter we have presented an admission policy and a neighbor search mechanism for in-network caching architectures. Our admission policy attempts to ensure only a single cached copy of a piece

of content, via informing other content routers via a bit in the header. The neighbor search strategy implements a cache cooperation protocol to exchange information about cached data between peer caches, allowing for content search in neighbor caches. Our key findings in the chapter are that an admission policy is needed for good performance and that neighbor search is beneficial in finding content in nearby CRs, thus avoiding the need to retrieve the content from the origin. We have implemented and evaluated the admission policy and neighbor search protocol and the results show an increase of 33% on the hit ratio and decrease of 23% and 20% on the average number of hops and footprint reduction. In addition, the proposed algorithm was compared with redundancy elimination thoroughly.

Chapter 5

Compact Routing in Hyperbolic Space

While Information-centric networking tries to solve many problems in the current Internet and opens the door to many novel applications, it also leaves many challenges unanswered, e.g., mobility support and mobile content publishing and dissemination. In this chapter, we use CCN as an example and show how a greedy routing can be implemented in CCN architecture to support mobility. This allows for efficient content publisher mobility and supports seamless handoffs for interactive connections. We present our solution – MobiCCN, and evaluate it thoroughly in realistic network topologies to show it outperforms other popular mobility schemes.

5.1 Introduction

Content-Centric Networking (CCN) [1] shifts the current Internet paradigm from point-to-point communication to content-centric dissemination. CCN aims at solving some problems in current Internet like network congestion, content delivery and security and etc. The essence of CCN is accessing content by name. All content is identified, addressed and retrieved by name instead of physical locations. A client requests a content item by sending out an `Interest` packet into the network. The network nodes can potentially store the content to serve the future requests.

Naming also poses significant challenges on routing. In current Internet, DNS handles the mapping between hostnames and IP addresses. However, as far as content is concerned, the number of items to handle is orders of magnitude larger than names handled by DNS. Hierarchical names have been proposed because they offer performance gains through aggregation.

However, this assumes the names are correlated with the underlying network topology, and may leave lots of gaps if the content has high mobility. While Caesar et al. showed flat name is feasible in [123], Ghodsi et al. argued in [23] that both hierarchical and flat names are essentially the same.

Due to its receiver-driven design, CCN inherently supports receiver mobility. A lost packet can be recovered by retransmitting the `Interest`. However, sender or publisher mobility is difficult to achieve. One scenario is maintaining the communication between mobile caller and callee in a VoIP call. Another is mobile content publishing and dissemination. In both cases, data sources are highly mobile. The standard CCN requires necessary name operations to handle data source mobility, but updating and propagating names are expensive operations in network. Performance and scalability are hard to achieve in such cases.

All these issues boil down to the question how we route the packets to the correct destination. Routing and name resolution form the core of the problem. An interesting direction in routing research is greedy routing which was first applied to mobile ad-hoc network, but it can also be used in traditional wired network. Zhang et al. in [124] presented a greedy routing based on underlying metric space. The idea is that each router is assigned a coordinate from a name-based metric space. The router makes the forwarding decisions based on the distance of its directly connected neighbors and the destination in the packet. CAIDA has also done experiments in evaluating greedy routing, but they have been limited to small-scale and manually assigned (geographical) coordinates [125].

In this chapter, we show that by introducing a greedy routing into CCN architecture, we can provide an efficient mechanism for seamless mobility, also solve the disparity between enormous space of names and scarce routers' resources. Even though we focus the discussion on CCN, it is worth pointing out that our solution can be equally applied to other similar systems. Our contributions are as follows:

1. We show that greedy routing can be implemented as routing policy in CCN.
2. We present MobiCCN, our mobility scheme, and evaluate it thoroughly in realistic settings.
3. We compare MobiCCN with other schemes from literature, and show that it outperforms them.

The chapter is organized as follows. We present MobiCCN in Section 5.2 and evaluate it in Section 5.3. Section 5.4 gives a comparison of common mobility schemes in CCN. Finally, Section 5.5 concludes the chapter.

5.2 System Model

In this section, we first give a brief introduction on CCN mechanisms and greedy routing technique. Then we present our MobiCCN design by describing how communication happens in the system. Furthermore, we give a specific scenario of VoIP in Section 5.2.

CCN Routing

Information-Centric Networking (ICN) has been an active research area for several years. There are several similar independent proposals [1–3], and all of them are essentially based on the similar concept – accessing content by name. We use CCN [1] as basis for our work, but many of the key elements of our solution can be applied in other ICN architectures as well.

CCN is a receiver-driven model. To retrieve a content, the user needs to construct an `Interest` packet first, which contains the content name. The `Interest` is sent to the network and routed in a hop-by-hop manner by CCN routers. Each router checks the content name and if there is a copy in the local storage – *Content Store* (`CS`), the response will be sent back immediately. Otherwise, the router checks its *Forwarding Information Base* (`FIB`) and uses longest prefix matching to determine which `face` the `Interest` will be forwarded to. The forwarded `Interest` leaves an entry in router's *Pending Interest Table* (`PIT`). If the `Interest` finally reaches the data source and is replied by the server, the response can follow the entries in `PIT` left by the previous `Interest` and goes back to the user.

CCN inherently supports receiver-side mobility. To recover a lost packet during mobility, receiver only needs to retransmit the previous `Interest`. Intermediate routers can use the copies in their `CS` to serve the request. However, if the data source is mobile, retrieving data will be much more difficult. Common schemes use hierarchical naming and if a data source moves into a new domain, it has to perform expensive name operations to handle the mobility. In some schemes, the receiver needs to be informed about the changes so that communication can continue. In [126], Kim et al. compared several mobility schemes, including their own solution.

Greedy Routing

Greedy routing has a long history in mobile and sensor networks. In such networks, a node does not have global knowledge of the network topology and only knows its neighbors. Greedy routing makes it possible to route packets in the "dark" by assigning coordinates to nodes. However, greedy

routing doesn't specify its underlying metric space, node usually uses its actual geographical coordinates as its locator, which is also referred as geographical routing. The destination's coordinate is embedded into the packet header. To forward a packet, a node calculates the distance between the destination and each of its directly connected neighbors, and selects the one closest to the destination as the next hop. However, using geographical coordinates cannot guarantee 100% delivery due to the *local minimum* issue. In a connected graph, *local minimum* refers to the situation that a node x itself is closer to the destination y than any of its neighbors even though y is not x's directly connected neighbor. In this case, the node cannot decide who should be the next hop therefore routing fails. In [127], Cvetkovski et al. proposed Gravity-Pressure routing to provide optional path when local minimum occurs.

Using geographical coordinates is equivalent to embedding the network into Euclidean space. However, Euclidean space is not the only candidate for greedy routing's underlying metric space. Instead of real geographical coordinates in Euclidean space, nodes can use virtual coordinates from any well-defined metric space. Therefore, another solution to the local minimum problem is to embed the topology in a "better" metric space. The idea is to find a *greedy embedding* for arbitrary topologies. Greedy embedding refers to the embedding with the property that given any destination y which is not directly connected to a node x, x can always find a neighbor of him who is closer to y than himself. Kleinberg gave a proof in [128], showing that if we use a hyperbolic space as the underlying metric space, then every connected graph has a greedy embedding. Therefore 100% delivery is guaranteed. In [127], Cvetkovski et al. extended Kleinberg's work to dynamic graphs.

Proposal

As we have seen, the conventional way of routing and name resolution in CCN is incapable of handling sender mobility issues. We propose a new routing protocol which can coexist with the standard CCN routing protocol.

There are two routing protocols in MobiCCN, the standard CCN protocol and the greedy protocol. MobiCCN neither changes the existing packet format nor adds any new ones. A greedy packet is just a normal CCN `Interest`. We only reserved prefix *greedy:/* for greedy protocol, while the standard one uses *ccnx:/*. Whenever a router receives a packet with the name starting with *greedy:/*, it switches to greedy protocol to forward the packet. As Figure 5.1 shows, the general format of the name of a greedy packet is *greedy:/vc/operation/parameters/....*

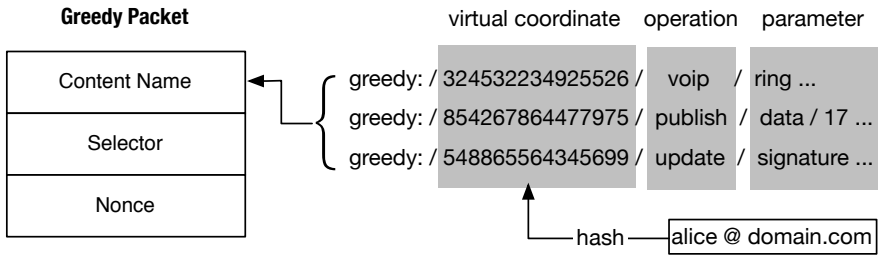


Figure 5.1: Greedy packet type. All the greedy packets are normal CCN `Interests`, MobiCCN only reserves the prefix *greedy:/* to activate the greedy protocol.

Each router is assigned a *vc (virtual coordinate)* from the underlying hyperbolic space $\mathbb{H}$ (using MWST algorithm discussed in Section 5.3). Coordinate allocation can be done in many ways, e.g., manually or by the ISP using the MWST algorithm (or any similar embedding algorithm). As shown in [127], coordinate allocation can also be done automatically and dynamically whenever a new node joins or an old node leaves by using the dynamic embedding algorithm in [127].

Greedy protocol uses *vc* as the destination address and embeds it into the content name of a packet. Each router only maintains a small table of its neighbors' coordinates. In order to forward a packet, the router first extracts destination coordinate *vc* from the packet, then it calculates the distance between the destination and each of its neighbors. The packet is forwarded to the neighbor who is closest to the destination.

However, we do not have to calculate distance for every greedy packet and can reduce the computational overheads by caching previous results. When a greedy packet arrives, the router checks whether there is an entry in `FIB` using the longest prefix matching. If the result is positive, it means the distance has been calculated before, then packet is forwarded to the next hop stored in `FIB`. The router's performance is independent of the number of greedy packets passing by.

Each user has a unique ID to identify himself, which can be the same as his CCN name. Greedy routing then maps the user ID into the same hyperbolic space $\mathbb{H}$ to obtain its virtual coordinate. Mapping can be done with any well-defined hash function, e.g. SHA-1. Each user has a dedicated router who is closest to him in $\mathbb{H}$ as his *host router* in the network. The host router serves as rendezvous point and relays traffic for him. [1]

[1]There are obvious similarities between MobiCCN and Mobile IP, and the host router

1: **Input:** Greedy Interest GI
2: **Output:** Forward decision
3: $dst \leftarrow$ destination coordinate of GI
4: **if** $(dst, face)$ in FIB **then**
5: $oface \leftarrow face$
6: **else**
7: **for** each directly connected neighbor N_i **do**
8: $d_i \leftarrow$ distance between dst and N_i
9: **end for**
10: $oface \leftarrow N_i$ with smallest d_i
11: Update $(dst, oface)$ pair in FIB
12: **end if**
13: Forward GI to $oface$

Algorithm 2: Greedy routing protocol - Interest

Whenever a data source moves to a new attachment point, it sends out an `Update` packet to its host router. The `Update` has a name like *greedy:/vc/update/...*, indicating it is an update operation. Each router the `Update` passes by will update the corresponding entries of that data source in its `FIB` accordingly; then `Interests` towards the source can be forwarded correctly to the new domain. From receiver's perspective, it always uses data source's original name to communicate. So there is no need to change content name even after the source's handoff, because the `Interests` can always reach source's host router (in the worst case).

Algorithms 2 and 3 show how greedy `Interest` and greedy `Update` are processed in a router. It is not necessary for a greedy `Interest` to arrive at the rendezvous point in order to reach the mobile user, because the greedy `Interest` may pass a router which already cached $(dst, face)$ in `FIB` before reaching the rendezvous point. This means that the stretch in MobiCCN can even be lower than that in normal greedy routing, however this advantage is highly topology-dependent and not a guaranteed property.

Security

As shown in [1], CCN is built on the notion of *content-based security*. Each piece of content can be authenticated by the digital signature embedded in the packet header. MobiCCN inherently embodies the CCN's security, and has the equal strength of CCN in against many kinds of attacks.

is roughly equivalent to a home agent.

1: **Input:** Greedy Update GU
2: **Output:** Update FIB of the routers between a mobile user and its corresponding rendezvous point
3: $iface \leftarrow$ ingress of GU
4: $dst \leftarrow$ destination coordinate of GU
5: Update $(dst, iface)$ pair in FIB
6: **for** each directly connected neighbor N_i **do**
7: $d_i \leftarrow$ distance between dst and N_i
8: **end for**
9: $oface \leftarrow N_i$ with smallest d_i
10: $d \leftarrow$ destination between dst and current router
11: **if** $d < min(d_i)$ **then**
12: Rendezvous point, stop forwarding
13: **else**
14: Forward GU to $oface$
15: **end if**

Algorithm 3: Greedy routing protocol - Update

To prevent malicious users from exploiting `Update` packets to disturb the normal communication, the sender is required to sign every `Update` packet. Whenever an `Update` packet arrives, the router needs to check whether the sender is the actual owner of the name so that he has the right to update his corresponding entries in `FIB`. This can be easily done by validating the digital signature in the packet, if the key used for signing is not the same as the one of the actual owner, the packet should be dropped immediately. Technically, this means signed `Interest` instead of signed Content Object. According to CCNx specification, the signature can either be appended to the content name (as MobiCCN does), or stored in the additional field in the header.

Validation requires extra operations and network communication. To reduce the overheads and also prevent work-factor attack, we adopted three mechanisms in the design. First, router is not obligated to validate every `Update` packet as long as sender's attachment point remains unchanged. Second, only the edge router (sender's attachment point) is responsible for validating sender's authenticity, the downstream routers are free from these operations. Third, owner's key is cached before its expiration so that a router needs not retrieve it all the time whenever it needs validation.

Instead of mandating a *one-size-fits-all* approach, CCN adopts a very flexible *contextual* trust model. Clients should determine themselves whether the data is trustworthy. In a similar vein, MobiCCN design allows both tra-

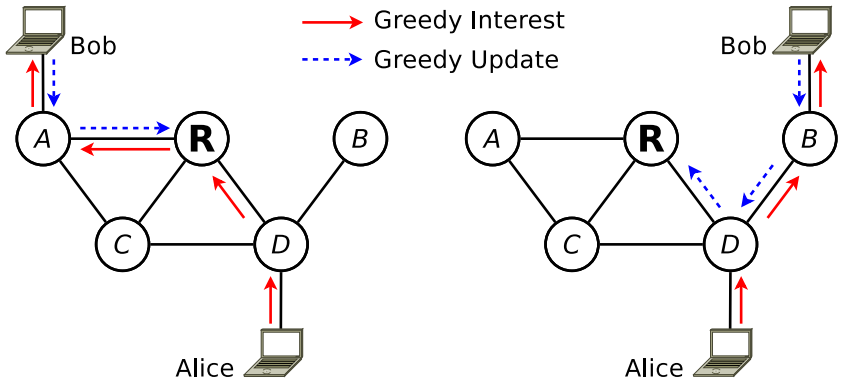


Figure 5.2: MobiCCN Scenario. Bob's host router is R and between the left and right figure Bob changes his attachment point from A to B. Router D caches Bob's `Update` and Alice's `Interest` packet on the right-hand figure does not need to go to R but D is able to forward it directly to B.

ditional hierarchical **PKI** (public-key infrastructure) and non-hierarchical model like **SDSI** [129]. However, discussion over the trust models is beyond the scope of this chapter and is left as our future work.

VoIP Scenario

We present a VoIP scenario as an example of how MobiCCN handles mobility. Note that MobiCCN is not limited to voice calls, but is a solution for general publisher mobility in CCN.

Figure 5.2 shows a simple VoIP scenario in MobiCCN with both two users – Alice (Caller) and Bob (Callee). Both Alice and Bob use their email addresses as their unique IDs, and their corresponding virtual coordinates in $\mathbb{H}$ are vc_{alice} and vc_{bob} respectively. Now Alice wants to initialize a VoIP call to Bob. To set up the connection, she first tries the standard CCN way by using the name like *ccnx:/domain/voip/bob*. However, if Bob already starts moving before Alice's attempts, the connection setup will fail. After the timeout event is triggered, Alice activates greedy protocol and sends out a new `Interest` with the name *greedy:/*vc_{bob}*/voip/ring*. The `Interest` will pass by Bob's host router and finally find its way to Bob using greedy protocol. If both fail, then Bob is considered offline.

The previous two-attempt setup only happens once in the beginning of the communication, and the overhead can be avoided if greedy protocol is used as the primary protocol. The standard one is then used as backup protocol to achieve lower average latency if both counterparts are station-

ary.

Another way to avoid two-attempt setup is for Alice to send out two `Interests` (both standard and greedy one) in parallel in the beginning, then choose the protocol after Bob replies. The overhead is just one extra packet.

When Bob is the data source and he moves into a new domain B, he sends a greedy `Update` with the name *greedy:/vc$_{bob}$/update* after the handoff. The `Update` eventually reaches his host router and all the intermediate routers update their `FIB` in order to forward `Interests` towards Bob correctly.

However, as we can see from Figure 5.2, Alice's `Interest` reaches D before R, and router D already updated its `FIB` from Bob's greedy `Update`. Alice's `Interest` can then be routed directly towards Bob without passing the host router.

Features & Limitations

MobiCCN can either be used as a backup solution for mobility issues, or the primary scheme for mobile content publishing and dissemination. MobiCCN only needs marginal modifications to CCN routers and does not interfere with the standard protocol. Applications using the standard protocol are not aware of the greedy protocol.

Greedy protocol might increase stretch in the communication. However, as we show in Section 5.4, it is still much lower than other popular schemes and stretch can be further reduced by using a better embedding algorithm. Furthermore, due to the flexibility of MobiCCN, users can negotiate with each other to switch to the standard protocol if they stop moving.

5.3 Evaluation in Mobility Scenarios

Prototype & Testbed

We implemented MobiCCN prototype in Python. The prototype works similarly to CCN defined in [1]. Greedy routing is implemented as an extension to the standard CCNx routing protocol. We are also implementing MobiCCN in the CCNx prototype as a plug-in.

We chose four real-world ISPs networks to run our experiments: Exodus, Sprint, AT&T and NTT. The network topology files are from Rocketfuel [84] project. For the network with multiple components, we only use the biggest one. Table 5.1 shows an overview of the networks with their graph properties.

Network	Routers	Links	POPs	Diameter	Avg. Path
Exodus	338	800	23	12	5.824
Sprint	547	1600	43	12	5.182
AT&T	733	2300	108	11	6.043
NTT	1018	2300	121	14	6.203

Table 5.1: Graph properties of the four selected ISP networks

All the experiments are performed on our department cluster consisting of 240 Dell PowerEdge M610 nodes. Each node is equipped with 2 quad-core CPUs, 32GB memory, and connected to a 10-Gbit network. All the nodes run Ubuntu SMP with 2.6.32 kernel. Multiple virtual routers are multiplexed onto one physical node if the ISPs network is larger than the cluster network.

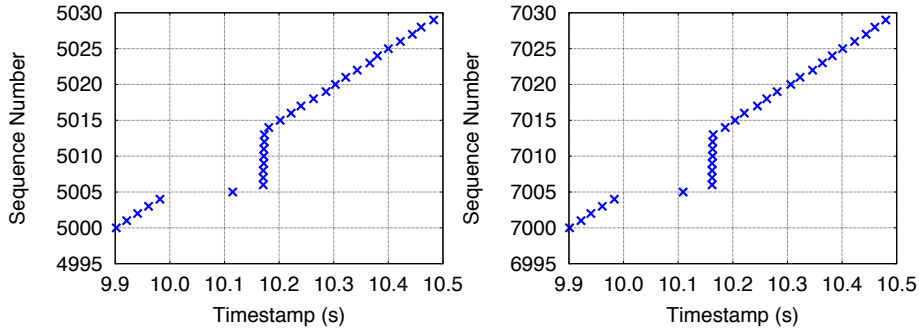
Handoff Delay

Handoff delay is one of the most important metrics for evaluating a mobility solution. We experimented our solution on four topologies, but since the results are similar in all of them, we only present the results on AT&T network. We also compared MobiCCN with different mobility schemes. However, since Interest Forwarding has been shown to be superior to the others [126], we only compare against it in this section. Note that the evaluation in [126] is done on a synthetic topology and we now run their algorithm on a real ISP topology.

In our simulation, the link delay is set to 5 ms. The initial placement of the sender and receiver is arbitrary. The selection of the next attachment point of the mobile sender is among the nodes within a 2-hop radius. Layer 2 handoff delay is set to 100 ms, and loss detection timer is also set to 100 ms. Both caller and callee perform a simultaneous handoff at 10 sec. Caller and callee send out `Interests` at a rate of 50 pkts/s.

Figure 5.3a shows the sequence number of the content piece the caller received when simultaneous handoff happened. When the caller finished layer 2 handoff at 10.1 sec, he started re-requesting the lost data. Because packet #5005 was already on its way to the caller when the handoff happened, it was already cached by an intermediate router. That is why packet #5005 can be quickly re-transmitted at 10.15 sec just after the layer 2 handoff finished. The rest of the re-transmissions are subject to one RTT, they arrive later at 10.17 sec. The caller's handoff delay is 173 ms.

Figure 5.3b shows sequence number callee received during the hand-



(a) Caller (b) Callee

Figure 5.3: MobiCCN handoff delay

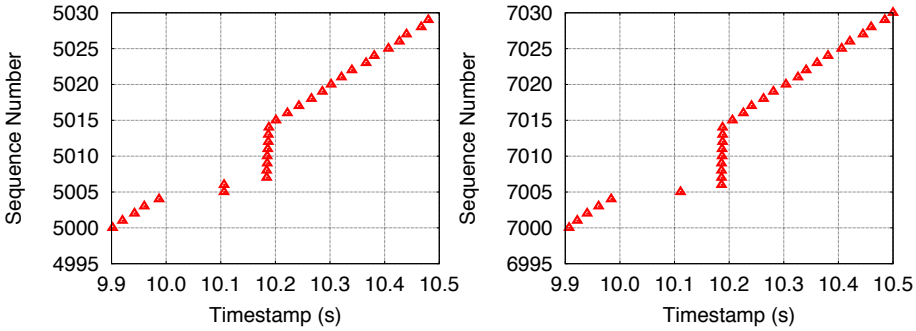
off. The callee's handoff delay is 163 ms, which is shorter than the caller's 173 ms. The reason is that paths between caller and callee are not symmetric. Path from callee to caller (6 hops) is shorter than that from caller to callee (7 hops).

Figure 5.4 shows the handoff delay in Interest forwarding scheme from [126]. The experiments are done with the same setting as that of MobiCCN. The caller's and callee's handoff delays are the same, both are 188 ms. Although small, this difference to MobiCCN is consistently present and measurable in all our experiments. The reason of the longer handoff delay is that the path between caller and callee increased from 6 hops to 8 hops after the handoff. This is shown in Figure 5.5. If data source moves from A to B, topology α in Figure 5.5a will not increase the path length. However, topology β in Figure 5.5b will increase the path length by 1. Triangular routing cannot be eliminated in Interest Forwarding if user's home agent A becomes the next hop in the new path. It is more difficult to prevent this issue if topology β is closer to the network core.

Even though neither MobiCCN nor Interest Forwarding requires users to change names after the handoff, Interest Forwarding may be affected by the network topology.

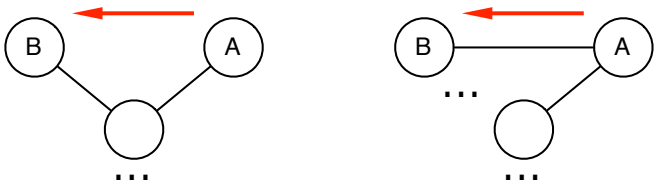
Scalability

We designed another experiment to see how the network topology affects the path stretch. In the experiment, callee is fixed and caller moves N times. Every t sec, caller moves to a new attachment point. (While such mobility might not happen in many scenarios, it serves to illustrate MobiCCN's scalability even under extremely mobility.) We measured the stretch be-



(a) Caller (b) Callee

Figure 5.4: Interest Forwarding [126] handoff delay



(a) Topology α (b) Topology β

Figure 5.5: Interest Forwarding is subject to topology

tween caller and callee after each handoff. The experiments were repeated 50 times, and Figure 5.6 shows the average stretch.

For Interest Forwarding, despite of some small fluctuations, stretch increases while the caller keeps changing its attachment point. The reason is that if the previously attached router is the next hop of the newly attached router, the path will increase after handoff. As the caller moves more, the probability of this happening varies according to the topology, thus causing some fluctuation in the results. However stretch shows a steadily increasing trend. Furthermore, stretch in Interest Forwarding scheme is consistently higher than in MobiCCN, which is stable at 1.13. MobiCCN's performance is independent of moving and topology once the embedding is done.

This experiment implies the network topology can have a significant impact on the performance of a mobility scheme. An artificial topology is incapable of reflecting all the characteristics from realistic topologies, thus evaluations on purely synthetic topologies are likely to yield results that do not correspond to results in a real network topology.

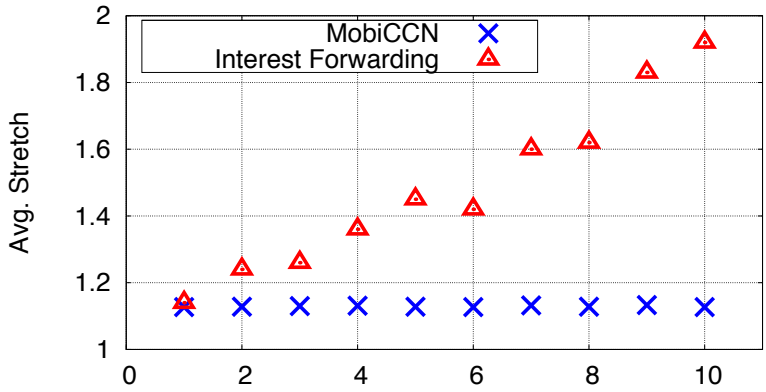


Figure 5.6: Average stretch as a function of number of handoffs

Stretch

The host router approach of MobiCCN may increase stretch because traffic in many cases passes through the host router, but a better embedding algorithm can help reduce the stretch.

To embed a network into a hyperbolic space, the first step is to derive a spanning tree from the network, then we embed the tree into the space. Kleinberg showed in [128] that the greedy embedding of a spanning tree of a graph is also the graph's greedy embedding. However we can derive multiple spanning trees from the same graph, and different trees may lead to different stretches. When the network is small, the embedding can be done manually and the stretch can be reduced to as low as 1, like [125]. However, manually assigning coordinates is infeasible for a large network.

In [130], Cvetkovski et al. implemented two heuristics and showed that they can improve the average hop stretch by about 30%. In this chapter, we used the Maximum-Weight Spanning Tree (MWST) in [130] to construct the spanning tree on the experiment topology and embedded it into a Poincaré disk.

For each network, we generated 5000 random minimum spanning trees and embedded them into the Poincaré disk. We recorded the average and the minimum value; these are shown in the first and second row in Table 5.2.

We used MWST for the greedy embedding and recorded its stretch and also calculated the improvement MWST achieved compared with the average value. The third and fourth row in Table 5.2 show the results. MWST has about 11% – 13% improvement on realistic network topologies.

	Exodus	**Sprint**	**AT&T**	**NTT**
Avg. Stretch	1.384	1.375	1.271	1.320
Min. Stretch	1.149	1.197	1.110	1.198
MWST	1.212	1.185	1.128	1.150
Improvement	13.06%	13.82%	11.25%	12.88%

Table 5.2: Stretch of four networks with different spanning tree algorithms

Performance Impact

When CCN router forwards a greedy packet, router spends extra CPU cycles in computing the distances between the destination and its neighbors to decide the next hop. However, since MobiCCN uses the same longest prefix matching mechanism as that in standard CCN, it can utilize `FIB` to cache the previously calculated results to reduce the overheads. Then the CPU overheads become independent of the absolute number of greedy packets passing by, but only a function of the arrival rate of the packets containing new destinations. The `FIB` entries are tentative and will be purged out automatically after predefined expiration time. So even if all the traffic are greedy packets, the overheads still remain at a low level.

We evaluated how greedy routing impacts router performance in the worst case without optimization. In our experiment, we first measured the router's maximum throughput when all the packets passed by are standard CCN packets. Then we increased the fraction of greedy packets step by step, and examined how it degrades CCN router's throughput. Our results show that the throughput drops linearly as a function of the fraction of greedy packets. With up to 10% greedy packets, the drop is negligible, but if the traffic consists of purely greedy packets and the router has to calculate the distance for every packet, the throughput drops by about 30%.

5.4 Comparison of Mobility Schemes

In this section, we compare MobiCCN with other mobility schemes presented in literature. We evaluate the schemes from the following perspectives: handoff delay, average latency, capability of handling simultaneous handoff, scalability, single point of failure and implementation complexity. Table 5.3 summarizes the comparison.

	Avg. Latency	**Handoff Delay**	**Simultaneous Handoff**	**Scalability**	**Single Point of Failure**	**Complexity**
MobiCCN	Medium	Low	Yes	High	No	Medium
Sender-Driven Msg	Low	High	No	High	No	Low
Rendezvous Point	Low	Medium	Yes	Low	Yes	Low
Indirection Point	High	Medium	Yes	Low	Yes	High
Interest Forwarding	Medium	Low	Yes	Medium	No	High

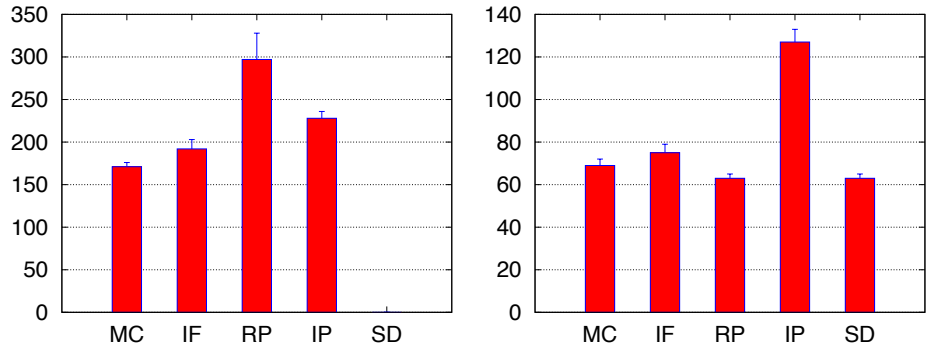
Table 5.3: Comparison of different mobility schemes. MobiCCN achieves good tradeoff point from various perspective.

Sender-driven control message is the most straightforward scheme. In this scheme, the moving user sends out a control message explicitly to the receiver to inform his new hierarchical name when handoff occurs. However, this scheme cannot handle well the situation where both sender and receiver are moving. The communication may completely break down when simultaneous handoff happens. Another problem is that receiver must regenerate the new `Interest` for the lost packet using sender's new hierarchical name. The advantages of this scheme are its simplicity and pure CCN style, and average latency in the communication is low. All the modifications are in the application layer.

In **Rendezvous point** scheme, user needs to update their attachment point to the rendezvous point periodically or when handoff occurs. If the receiver fails to get response within a predefined time, the receiver will think the data source has changed its attachment point. Then the receiver sends the query to rendezvous point to get the update. In this scheme, the communication will not completely break down if simultaneous handoff happens, but it is still possible that the receiver gets outdated information and suffers from a large delay due to second lookup operations. Furthermore, the receiver still needs to regenerate `Interest` with the new name for the lost packets. Generally, this scheme suffers from a large handoff delay. The advantage is that average latency is low and the modification is on the application layer; lookup only happens when timeout is triggered. Normal communication is done as in CCN.

Indirection point scheme uses separate server to relay all the traffic. If handoff occurs, the `Interests` to the user are buffered first at the indirection point, then forwarded later until the moving user updates the new name to the Indirection point. Because all the traffic must pass the indirection point, the obvious disadvantage is the indirection point becomes the single point of failure and a bottleneck if the traffic load is heavy. Even though the handoff delay can be improved in this scheme, normal traffic suffers from a large average latency.

In all aforementioned schemes, the content must change its name based on the attached domain. However, updating content name is an expensive operation in CCN and this makes seamless handoff difficult to implement. Kim et al. proposed **Interest Forwarding** in [126]. In their scheme, the mobile user must send a notification to the current attached router when it notices a handoff is imminent. The router will start buffering the coming `Interests` for the user. Then the user can fetch the buffered `Interests` by sending a `virtual Interest` back to the old attached router. The `virtual Interest` also updates the `FIB` in the intermediate routers so that



(a) Handoff delay (ms) (b) Average latency (ms)

Figure 5.7: Comparison of latencies in different schemes. MC: MobiCCN, IF: Interest Forwarding, RP: Rendezvous Point, IP: Indirection Point, SD: Sender-Driven Message

the following `Interests` can be forwarded correctly. This scheme avoids changing the content name by using tentative home agent. However, one problem is the whole scheme may fail if an imminent handoff becomes hard to predict. Secondly, as we have shown in this chapter, the path may grow longer while the user is moving, and the following traffic suffers from the larger latency.

Figure 5.7 shows the handoff delay and average latency in each scheme. The experiment was repeated 100 times and the average value with standard deviation is presented. In Figure 5.7a, simultaneous handoff was evaluated. Sender-Driven Scheme is missing because it cannot handle simultaneous handoff. The performance of Rendezvous Point and Indirection Point depends on the placement of the Indirection/Rendezvous server. In our experiment, we deployed the server 6 hops away from both two mobile nodes. MobiCCN has the best performance of all the solutions. Rendezvous Point is the worst and has the largest variation due to the possibility that user may receive outdated information.

Figure 5.7b shows the average latency in the communication. We let the data source have two handoffs before we start the evaluation. The Rendezvous Point and Sender-Driven Message have the shortest latency because they always use the shortest path. Indirection Point is the worst because all the traffic is relayed. MobiCCN is slightly higher than the best one due to the stretch caused by greedy routing. But the latency can be further reduced and become closer to the one by using better embedding algorithm. Interest Forwarding is a little higher than MobiCCN due to the

issue we discussed in Section 5.3.

In summary, MobiCCN outperforms the other solutions in terms of delay and (for the most part) latency.

5.5 Conclusion

In this chapter, we present how we extend geographical routing in current CCNx to solve mobility and mobile content publishing and dissemination issues in CCN. By embedding network topology into hyperbolic space, we distribute the rendezvous points and name resolution functionality into the network. We compared MobiCCN to other proposed CCN mobility schemes and showed that it outperforms existing schemes both in terms of handoff delay and communication latency. We are currently implementing our solution on CCNx as extension, which is fully compatible with the standard CCN routing protocol.

For the future work, we have three possible directions as follows: first, looking for better embedding algorithms with lower stretch and better adaptability to network dynamics; second, reserving CCN naming context under greedy protocol; third, evaluating the overheads in the realistic network environment.

Chapter 6

Prefix-S Embedding and Topology-Aware Hashing

In the previous chapter, we focused on mobility of the content source, using network embeddings as a tool for content addressing and mobility management. In this chapter, we first show though previously designed embedding is able to solve the mobility issue, it leads to a highly unbalanced storage and traffic load: more than 90 % of all stored references are mapped to one node, which is involved in more than 95% of all queries. Then, we propose a modified embedding, *Prefix-S embedding*, and a topology-aware key assignment, which enable a uniform distribution of the storage load. The maximum traffic per node is also considerably reduced from more than 95% to 35%.

6.1 Introduction

Compared to the web, most existing ICN proposals typically offer natural consumer mobility, since the system architectures behind the proposals are essentially receiver-driven. However, provider mobility, which usually requires expensive name operations (e.g., propagating updates by flooding or maintaining large databases) in the network, is extremely difficult to implement in an efficient and scalable way.

Thus, an ICN network is expected to either implement name resolution or content-based routing in order to discover and deliver content. The choice on these two approaches is a trade-off between efficiency and accuracy. Name resolution guarantees content discovery but has to maintain two databases (identifier-to-locator mapping and reachability information) at a logically centralized point and suffers from query overheads. Content-based

routing, on the other hand, is more robust and efficient since it bypasses the query step but only promises probabilistic content discovery (with tunable probability) and suffers from high traffic overheads. However, as [25] showed, this tradeoff can be mediated by using a flat naming scheme with greedy routing.

In technical terms, [25] uses SHA-1 for content addressing and hyperbolic embedding technique to implement a distributed hash table (DHT) underlay in the network, further transforming every content router into a rendezvous point. Note that there are multiple ways of addressing content and hyperbolic embedding with greedy routing is merely one possible realization of compact routing. Though [25] showed that their solution is superior to existing alternatives in terms of handling mobility, it fails to take into account the complexity of the embedding. As an extension to the work in [25], in this chapter we focus on looking for a superior substitute to hyperbolic embedding, and study how content addressing should take advantage of the underlying structure of the metric space in a routing scheme. We show that a poor choice of embedding can cause a highly skewed name distribution in routing's metric space, which further may lead to severe congestion. Our contributions in this chapter are as follows:

1. We study several embedding schemes and propose Prefix-S embedding as a superior substitute to hyperbolic embedding. Prefix-S embedding provides guaranteed delivery at lower computational costs.

2. We show that content addressing with naive hashing may cause a highly unbalanced load distribution, mapping most content on one node. We propose a topology-aware hashing which takes network topological properties into account to achieve a uniformly distributed storage load.

3. We evaluate our solution thoroughly with realistic settings and show it outperforms the existing ones and achieves good system performance in terms of low overheads, high scalability and well-balanced load.

The rest of the chapter is organized as follows: Section 6.2 describes our system model and Section 6.3 presents the Prefix-S and topology-aware hashing algorithms. Section 6.4 evaluates the proposed algorithms with different simulation settings and Section 6.5 concludes the chapter.

6.2 System Model

In our model, we consider an information-centric network using a flat naming scheme. While our evaluation focuses on a CCN-like network, our use of hashing of names means that the results and model apply to any ICN proposal. In the rest of the chapter, we follow CCN's terminology. The network topology can be represented as a graph $G = (V, E)$ where node set V corresponds to the content routers and edge set E corresponds to the links among the connected routers. Each router v_i has a unique ID $id(v_i)$ allocated from a well-defined metric space $< M, d >$ with distance function d. The assignment $id : V \rightarrow M$ is called an embedding. In general, we want id to enable greedy routing, i.e., a path between any two nodes in V can be found by always choosing the neighbor of the currently contacted node, whose ID minimizes the distance to the target ID. Note that the distance of nodes refers to the distance of their IDs, rather than their topological distance.

Each content piece c_j is uniquely identified with a name drawn from a name space C. We denote the addressing function which maps a content name into M as $g : C \rightarrow M$. A content c_j has a designated host v_i such that $v_i = \arg\min_{\forall v_i \in V} d(id(v_i), g(c_j))$. In other words, c_j is designated to the router with the closest ID v_i in the metric space M.

Our system operation model is based on the work in [25]. However, since our concern is how embedding and hashing can influence the ID distribution in the metric space, which further impact the system performance. We simplify their model and only focus on the two aforementioned functions id and g instead of delving into practical protocol designs.

When user requests content c_i, the content name is hashed into M by calling $g(c_i)$ and embedded into the Interest packet. The Interest traverses the network with greedy routing. Each router only maintains a small table of its neighbors' coordinates. In order to forward an Interest, the router extracts destination coordinate (content ID) from its header, then it calculates the distance between the destination and each of its neighbors. The Interest is forwarded to the neighbor whose ID is closest to the destination.

The operation model defined in [25] requires that the content owner continuously registers himself at the host router so that the coming Interests can trace back. However, in our simplified model, we do not distinguish between content and content owner, and simply assume the Interest will be satisfied after arriving at its host router.

6.3 Embedding and Hashing Algorithms

In this section, we first review existing embeddings used for comparison in Section 6.4 before detailing our approach. Greedy embeddings are in general created by embedding a spanning tree into a suitable metric space. All nodes enumerate their children. The root node of the spanning tree is assigned an ID, and then each child computes its ID based on the ID of its parent and the index given by the parent.

Kleinberg's Embedding

Kleinberg's embedding into hyperbolic space enables embedding an arbitrary finite graph into two-dimensional hyperbolic space. The maximal degree m in the spanning tree needs to be known beforehand. The spanning tree is embedded into the Poincaré Disk by choosing the ID of the root node as the center of the ideal m-gon whose corners are m-th roots of unity, i.e. the complex numbers $e^{2\pi ij/m}$ for $j = 0, \ldots, m-1$. The ID of a child is obtained by applying a suitable isometric transformation, parametrized by the ID of the parent and the index of the child, of the above m-gon and choosing the center as the child's ID. Greedy routing, which always selects the closest neighbor to a desired target ID, is guaranteed to succeed [128]. Kleinberg did not treat the issue of creating suitable routing keys for content addressing. The straight-forward solution is to obtain a two-dimensional coordinate $(r(f), \phi(f))$ of the content f in polar coordinates. More precisely, let h be a hash function with a z bit image. Then $r(f)$ and $\phi(f)$ can be chosen as $r(f) = \frac{h(f)}{2^z}$ and $\phi(f) = 2\pi\frac{h(f+1)}{2^z}$.

Prefix Embedding

Prefix Embedding is an adaption of the PIE embedding [131] for unweighted graphs that has been considered for content addressing in [132]. The idea is to assign IDs using a custom metric space such that the distance between node IDs is identical to their hop distance in the spanning tree. The root is given the empty vector. A child's ID is the ID of the parent and an additional coordinate equal to the index of the child. The distance between two IDs s and t is then given as the sum of the their lengths minus twice the common prefix length $cpl(s,t)$, so

$$d(s,t) = |s| + |t| - 2cpl(s,t). \tag{6.1}$$

Two solutions for content addressing have been proposed in [132]. We use virtual tree construction, which is more flexible and was indicated to

have slightly better load balancing properties. Here, the coordinates are bit sequences. For any node with c children, consider the maximally balanced binary tree with c leaves. The node can hence map each child to one leaf in the binary tree. The ID of the child is then the ID of the parent concatenated with bit sequence corresponding to the position of the leaf in the binary tree. Content addressing is done by hashing the content f with a z-bit hash function. The responsible node is then the node with the longest common prefix with $h(f)$. However, greedy routing need to be slightly modified because some internal nodes of the virtual binary tree do not exists. The distances of nodes represent the distances in the virtual tree rather than in the actual spanning tree, so that a node can appear closer to its sibling than its parent. Thus, if a node encounters a query for key and its ID is not a prefix of the key but it does not have any closer neighbor, the query has to be forwarded to the parent. The modified greedy routing algorithm is guaranteed to find the closest node [132].

Prefix-S Embedding

One problem with the above version of the Prefix embedding is that content is only stored at nodes with at most one child. We hence propose Prefix-S Embedding, which stores content at all nodes. Here, each node u is given two IDs, the routing ID $ID_R(u)$ and the storage ID $ID_S(u)$, a prefix of all keys stored at u. The embedding works very similarly to the virtual tree variant of Prefix Embedding. The root node gets assigned the empty vector as routing ID. When enumerating its children, an internal node u adds an additional virtual child u'. Routing IDs are assigned to the children by concatenating $ID_R(u)$ and the bit sequence of the corresponding leaf in the maximally balanced binary tree as described above. The storage ID of u is then the routing ID of the virtual node u', $ID_S(u) = ID_R(u')$. The pseudocode of Prefix-S Embedding is given in Algorithm 4, with symbol $||$ indicating concatenation operation.

As for Prefix Embedding, greedy routing with the modification of forwarding to a parent if the current node's storage ID is closest to the key but not a prefix is guaranteed to work. Since the storage ID always corresponds to a leaf node in the virtual tree, a node is responsible for a key if and only if its storage ID is a prefix of key (under the assumption that keys are longer than node IDs). If a node is not responsible for a key, the responsible node can be found by forwarding to the closest neighbor if said neighbor is closer, or by contacting the parent. The forwarding decision is described in Algorithm 5.

1: {Given: Graph G=(V,E) with spanning tree T}
2: {$children(v)$: children of node v in T, ||: concatenation}
3: Assign $ID_R(r) = ()$
4: Queue $q = \{r\}$
5: **while** q is not empty **do**
6: u = remove head of q
7: **if** $|children(u)| > 0$ **then**
8: Create balanced binary tree B of size $|children(u)| + 1$
9: Create mapping map: $children(u) \cup \{u\} \rightarrow leaves(B)$
10: **for** $v \in children(u)$ **do**
11: $ID_R(v) = ID_R(u)||map(v)$
12: add v to q
13: **end for**
14: $ID_S(u) = ID_R(u)||map(u)$
15: **else**
16: $ID_S(u) = ID_R(u)$
17: **end if**
18: **end while**

Algorithm 4: PrefixSEmbedding()

1: {Given: Graph G=(V,E), assignments ID_R, ID_S, spanning tree T}
2: {$parent(v)$: parent of node v in T}
3: { $N(v)$: neighbors of node v in G}
4: **if** $ID_S(u)$ is a prefix of key **then**
5: routing terminated
6: **end if**
7: $next = argmin\{d(ID_R(v), key) : v \in N(u)\}$
8: **if** $d(ID_S(u), key) < d(ID_R(next), key)$ **then**
9: return next
10: **else**
11: return parent(u)
12: **end if**

Algorithm 5: nextHopS(BitSequence key, Node u)

Topology-Aware Hashing Algorithm

For the above embedding, leaves on the same level of the virtual tree are responsible for the same fraction of IDs. Hence, for arbitrary unbalanced trees, the storage load is expected to be unbalanced. In the following, we discuss how to create topology-aware keys, which achieve a uniform distribution over of keys over nodes for arbitrary topologies. The principal idea is that the probability that a key with prefix x has prefix $x||0$ is approximately equal to the ratio of the leaves in the left subtree rooted at node x in the virtual tree and all leaves in the subtree rooted at x. We thus compute the key of a piece of content f iteratively, as detailed in Algorithm 6. The key is a bit sequence $b_1 b_2 ... b_z$, where z is larger than the depth of the spanning tree. Assume for all possible prefixes $d_i = b_1 ... b_i$, the number of nodes v for which d_i is a prefix of $ID(v)$ is known. In case of Prefix-S Embedding, we consider the storage ID $ID_S(v)$. A hash function h is chosen with image space 2^z for some $z \in \mathbb{N}$. The values $h_i = h(f \oplus i)$ need to be known for $i = 0 \dots depth(T)$ to locate the responsible node. The $(i+1)$-th digit b_{i+1} of the file ID d is computed on basis of d_i. The recursion anchor is given by the empty string d_0. Then we have

$$b_{i+1} = \begin{cases} 0, \frac{h_{i+1}}{2^z} \leq \frac{|\{v \in V : cpl(ID(v), d_i||0) = i+1\}|}{|\{v \in V : cpl(ID(v), d) = i\}|} \\ 1, otherwise \end{cases} \tag{6.2}$$

with $cpl(s,t)$ denoting the common prefix length. The process continues until the set $|\{v \in V : cpl(ID(v), d) = i\}|$ is empty, so that the responsible node is uniquely identified by the key d_i. In order to distinguish different pieces of content stored at the same node and have keys of identical length, we concatenate d_i with the last $z - i$ bits of $h(f)$.

6.4 Load Balance in Storage and Traffic

We evaluated the approaches proposed in Section 6.3 using 9 different topologies of autonomous systems (AS) and three metrics as follows.

i) the fraction of stored items each node is responsible for,

ii) the traffic distribution, i.e., fraction of queries processed by each node,

iii) the number of hops needed to discover the destination of the query using greedy routing.

We evaluate the correlation between i) and ii) to see if a high storage load might be partly compensated by experiencing less traffic and vice

1: {Given: Graph G=(V,E), assignment ID}
2: {*cpl*: common prefix length, ||: concatenation}
3: $\{s[i \dots j]$: bits i to j of $s\}$
4: $i = 0$
5: $key = ''$
6: **while** length(key) ¡ z **do**
7: **if** $|\{v \in V : cpl(ID(v), key) = i\}| = 0$ **then**
8: $key = key||h(f)[i+1, \dots, z]$
9: **else**
10: $hash = h(f \oplus i)/2^z$
11: **if** $hash \leq \frac{|\{v \in V : cpl(ID(v), key||0) = i+1\}|}{|\{v \in V : cpl(ID(v), key) = i\}|}$ **then**
12: $key = key||0$
13: **else**
14: $key = key||1$
15: **end if**
16: **end if**
17: **end while**
18: return key

Algorithm 6: ComputeKey(BitSequence f)

versa. Ranking the nodes by i) and ii) provided an overview of how storage and traffic is balanced between the nodes. The evaluation was conducted as follows: We first created a spanning tree of the graph executing a breadth-first search starting a random node. Then we generated a set of $k = 10,000$ random ASCII character strings of length 20, which we used to represent the queried content. Afterwards, we computed the embedding for all considered embedding algorithms. For each embedding and each applicable key generation scheme, we then created the keys from the character strings and executed a query for each key from a random start node. Hence, a total of 5 combinations of embedding and key generation were evaluated : The Kleinberg embedding KB with hashing into the unit disk as well as Prefix P_H/P_{TA} and Prefix-S Embedding PS_H/PS_{TA} using both straight-forward hashing (H) and topology-aware (TA) keys. The relation between the storage and the traffic load on nodes is measured by the Pearson correlation coefficient. Results were averaged over 20 runs.

The hop count iii) does not necessarily correspond to the stretch, which is defined as the average ratio of the length of the routing path to the shortest path for all pairs of nodes. Since iii) considers queries, it is lower than the stretch if nodes that are easily discovered, e.g., the root, receive a disproportional high number of queries. The paths actually traversed during routing are more important for the working system, so that we

choose this metric rather than the stretch.

Expectations

We expect the Kleinberg embedding to produce a very unbalanced load distribution. The root node is responsible for the majority of the unit disk when considering Euclidean space (which the hashing does) regardless of the structure of the tree. Similarly, the tree for Prefix and Prefix-S Embedding is bound to have leaves at very different levels, leading to a high storage load on those on a high level when using straight-forward hashing. The maximum storage load is likely to be even higher for Prefix-S Embedding because the virtual nodes corresponding to the storage ID of the internal nodes on the top levels are always leaves. However, for topology-aware keys the load is supposed to be uniformly distributed over all nodes (Prefix-S Embedding) or all leaf nodes (Prefix Embedding). When considering the traffic a node has to process rather than the storage load, we expect nodes on the higher levels to experience a higher load. It is not required that messages between nodes in different branches pass their common ancestor in the tree, since greedy routing also uses non-tree edges. Nevertheless, tree edges are more likely to be used, so that we expect an unbalanced traffic distribution for all spanning-tree based embeddings. Thus, there should also be a high positive correlation between traffic and storage load for the Kleinberg and Prefix-S embedding with standard hashing. Both allocate the majority of the queries and the traffic to the higher levels of the tree. When using topology-aware keys, the traffic should be uncorrelated to the uniformly distributed load for Prefix-S embedding. Prefix embedding allocates all files on leave nodes. These are rarely intermediate nodes for queries, however they frequently are destinations, so that the sign of the correlation is not immediately clear.

Previous work on greedy embeddings showed that they exhibit similarly short routes and a low stretch [127, 128, 132]. Potentially, the average routing length is slightly lower for Kleinberg's embedding due to high fraction of queries addressed to the root node, which is fast to route to using tree edges.

Results

We first consider the maximum load per node and the total traffic, for the 9 considered ASs. Afterwards, we analyze the distribution of the load for one exemplary AS, AS1239. In order to show the general applicability of our results, Table 6.1 summarizes the maximal storage and traffic load for

the 9 sample ASs. In addition, the average routing length is given in order to estimate delays and the overall traffic.

Maximum Load

For the Kleinberg embedding, the storage load is always above 90%, and the fraction of traffic the most loaded node has to process is above 95%. For Prefix and Prefix-S embedding with hashing, the highest storage load and traffic are between 20% to 40%, and 50% to 85%, respectively. The maximum load is slightly higher for Prefix-S because internal nodes on high levels receive a large fraction of queries. For topology-aware keys the maximum storage load is always less than twice the average load for Prefix-S embedding, the actual load depending on the size of the AS. For Prefix embedding, the maximum load is slightly higher, because only a subset of the nodes participate in storing. The maximum traffic is drastically reduced by topology-aware keys as well, being at most 35% and 65%. Here, Prefix-S embedding has no overall advantage over Prefix embedding.

Routing Length

We also analyzed if load balancing increased the overall traffic, i.e., the number of hops needed to resolve a query. Table 6.1 indicates that indeed the topology-aware keys exhibit a slightly longer routing length than with Kleinberg's embedding, but the difference is mostly around half a hop in average, and at most 0.76 hops (comparing Kleinberg KB and Prefix-S PS_{TA} for AS3967). Note that the difference was not only due to an increased stretch, since both the standard hashing and the topology-aware keys use the same topology. Rather, the reason seems to be the location of the nodes that are responsible for the queries. Prefix embedding, storing all content at leaves, in general showed the longest routes, whereas Kleinberg embedding, storing most of the content at the root, is potentially the least costly. Due to the tree structured, the shortest path to the route is found, but non-optimal routes are common for leave nodes.

We now focus on a single AS 1239 for further analysis, but emphasize that the results applied equally to the other ASes as well, but they have been excluded due to space limitations.

Storage Load Distribution

The distribution of the storage load is displayed in Figure 6.1a, using a cumulative distribution function (cdf) to show the fraction of files the k nodes with the highest load are responsible for. The curve shows a very

steep initial increase for Kleinberg's embedding as well as for the two Prefix embeddings with a standard content addressing. By introducing topology-aware keys, the storage load is balanced uniformly, so that the increase is close to linear. The curve for Prefix embedding with topology-aware keys has a steeper slope because internal nodes with more than one child do not receive any load, whereas for Prefix-S Embedding, the load is uniformly distributed between all nodes.

Traffic Distribution

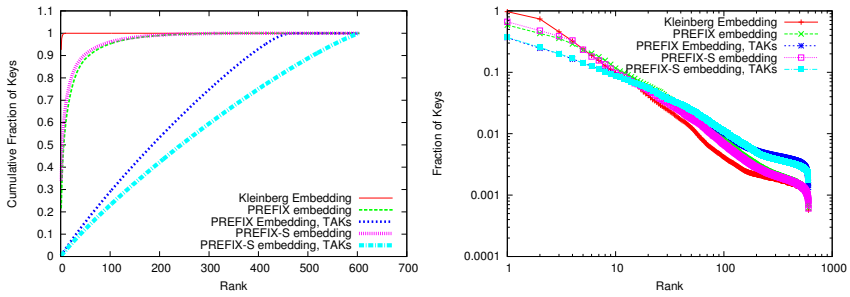
For the fraction of queries a node has to forward, i.e. the traffic per node, topology-aware keys also lessen the imbalance, but cannot abolish it (see Figure 6.1b). The nodes with the highest load are involved in more 35 % of all queries, which is however considerably less than for hyperbolic embeddings, for which the root node is involved in more than 98 % of the queries.

Correlation

As can be expected from these results, the correlation coefficients of the storage and the traffic load are high for Kleinberg (0.704) and Prefix-S embedding (0.629), whereas there is no notable correlation for Prefix-S embedding with topology-aware keys (0.011). For Prefix Embedding is correlation coefficient is clearly positive (0.316) for straight-forward hashing and clearly negative (-0.348) for topology-aware keys. The result can be explained since leaves nodes at a high level are frequent destinations of queries as well as storing a large number of items for the standard addressing scheme, leading to a positive correlation. For topology-aware keys, items are still stored only on leaf nodes, but uniformly distributed between them. Hence none of them has a disproportionally large amount of traffic, which is reserved for the internal nodes without storage responsibility, leading to a negative correlation.

Discussion

We have shown that the poor load balance of hyperbolic embeddings can be improved. Topology-aware keys achieve a uniform storage load and reduce the traffic at congested nodes at the price of a marginally increased overall traffic. The above results indicate that Prefix-S Embedding is the best choice when combined with topology-aware keys, since it offers a uniform storage distribution over all nodes and the lowest maximum traffic. However, Prefix Embedding offers a negative correlation between storage



(a) Storage (b) Traffic

Figure 6.1: Load Distribution: a) CDF of storage by rank, b) fraction of traffic ranked

and traffic, so that congested nodes only have to forward queries rather than answer them. Depending on the actual scenario, in particular storage and time constraints, Prefix embedding can be a better choice.

6.5 Conclusion

In this chapter, we studied the relation between the graph embedding and content addressing in the context of CCN. We showed that naive combination of the both is not only a costly solution but also causes highly skewed ID distribution in routing's metric space, further leading to storage load and congestion issues. To get around these issues, we proposed Prefix-S embedding to reduce the overhead and distribute the load among all nodes, and topology-aware keys to distribute the load uniformly. We evaluated our solution thoroughly and showed it outperforms others in realistic settings.

	Maximum Storage Load					Maximum Traffic					Average Routing Length				
AS	KB	P_H	P_{TA}	PS_H	PS_{TA}	KB	P_H	P_{TA}	PS_H	PS_{TA}	KB	P_H	P_{TA}	PS_H	PS_{TA}
1221	0.952	0.410	0.081	0.401	0.011	0.970	0.804	0.515	0.829	0.648	5.54	5.59	5.35	5.51	4.88
1239	0.925	0.216	0.004	0.340	0.003	0.972	0.588	0.367	0.665	0.369	4.94	5.41	5.00	5.39	5.28
1755	0.948	0.280	0.016	0.336	0.008	0.962	0.640	0.486	0.719	0.498	5.25	5.52	5.93	5.31	5.57
2914	0.950	0.270	0.003	0.289	0.002	0.972	0.699	0.348	0.763	0.350	5.90	6.22	6.36	6.01	6.20
3257	0.950	0.337	0.010	0.375	0.006	0.975	0.813	0.562	0.841	0.574	5.30	5.81	6.03	5.45	5.84
3356	0.950	0.185	0.004	0.228	0.003	0.973	0.507	0.351	0.582	0.361	3.84	4.38	4.26	4.23	4.18
3967	0.943	0.270	0.010	0.301	0.007	0.963	0.567	0.434	0.627	0.428	5.05	5.39	5.98	5.11	5.81
6461	0.922	0.351	0.117	0.389	0.052	0.953	0.638	0.575	0.698	0.567	3.34	3.55	3.57	3.42	3.41
7018	0.927	0.238	0.004	0.286	0.003	0.973	0.597	0.401	0.648	0.379	5.60	5.68	6.27	5.50	6.23

Table 6.1: Maximum load and routing length for various AS topologies with the following embedding/content addressing schemes: KB-Kleinberg Embedding, P_H-Prefix Embedding with standard hashing,P_{TA}: Prefix Embedding with topology-aware keys, PS_H-Prefix-S Embedding with standard hashing,PS_{TA}: Prefix-S Embedding with topology-aware keys

Chapter 7

Effects of Cooperation Policy on General Topologies

In this chapter we show that there is a tradeoff between two common caching metrics, byte hit rate and footprint reduction, and show that a cooperation policy can adjust this tradeoff. We model the cooperation policy with two parameters – search radius r and number of copies in the network K. These two parameters represent the range of cooperation and tolerance of duplicates. We show how cooperation impacts content distribution, and further illustrate the relation between content popularity and topological properties. Our work leads many implications on how to take advantage of topological properties in in-network caching strategy design.

7.1 Introduction

Caching is a key component in information-centric networks (ICN) [1–3, 9]. In-network caching not only reduces an ISP's outgoing traffic, but also reduces traffic within an ISP network. Byte hit rate (BHR) is a common metric for evaluating savings in inter-ISP traffic and footprint reduction (FPR) [111] measures the amount of traffic eliminated by caching (as product of traffic volume and distance it travels in the network) and is the metric we use for intra-ISP traffic savings. We show that BHR by itself is insufficient in capturing the performance of a network of caches; this is often overlooked by existing work. This chapter shows that there is a subtle interplay between BHR and FPR and that in some cases these two metrics oppose each other. We argue that a *cooperation policy* among routers can mediate this tradeoff between BHR and FPR and show that two parameters can tune the desired operating region: maximum number of duplicates

for each content item (K) and the radius for cooperation (r).

To improve BHR, a cooperation policy covering a large radius enhances network storage utilization by reducing the number of duplicates (*cache diversity* in [133]). However, large-scale cooperation causes communication overheads and increases intra-ISP traffic because requests may be redirected many times. Despite efforts in designing a cooperation policy [43,74,134], a proper model for its impact on BHR and FPR is still missing. We characterize cooperation policy by its search strength (r) and capability of reducing duplicates (K). We show how different r and K values lead to different tradeoffs between BHR and FPR, and discuss their implication.

There is considerable interest in exploiting topological properties in cache networks. Initial efforts [79, 80] indicate centrality as a promising metric, but questions like how to measure the topological impact on performance and mechanism of the interplay between topology, and caching strategy still remain open. We use a cache cooperation policy to couple content with topology and show that this coupling explains how topological properties impact caching performance; the tightness of the coupling indicates degree of topology's impact.

Our contributions in this chapter are as follows:

1. We highlight the importance of FPR as a performance metric for in-network caching, and show how BHR and FPR conflict each other at the Pareto frontier.

2. We propose a cooperation policy model to show the relationship between cooperation policy, content, and topology. We also categorize different cooperation types.

3. We propose a novel way to measure the impact of topology, and perform a thorough numerical analysis to show how it influences system performance.

7.2 System Model

Consider a network of M routers, L of which directly receive user requests and are *edge routers*. A router denoted by R_i is equipped with a storage capacity of C_i bytes. We assume N distinct files, denoted by f_i and being s_i bytes in size. All files are stored permanently at the Content Provider (CP) represented as the $(M+1)^{\text{th}}$ router (R_{M+1}). Denote the request probability of f_i by this file's *popularity* p_i, and denote the popularity vector by $\mathbf{p} = [p_i]$. When a request for f_i arrives to an edge router R_j ($j = \{1, ..., L\}$), R_j first

searches f_i in its cache. If R_j possesses it, R_j transmits f_i to the user; this a *hit*. Otherwise, in case of a *miss*, R_j contacts routers in its *r-hop neighborhood* to see if any of them has f_i; this is the *cooperation policy*. We call the set of all routers located at most r-hops away from R_j as the *searchable set* of R_j, denoted by $\mathcal{S}_j^c$. If f_i is stored in $\mathcal{S}_j^c$, it is retrieved to R_j from the closest router (if multiple routers holding f_i) and forwarded to the user. Let $\mathcal{R}_{j,CP}$ be the set of all routers on the path between a edge router R_j and the CP (excluding the CP). If no router in $\mathcal{S}_j^c$ stores the item, the request is routed to the next router in $\mathcal{R}_{j,CP}$ and searched there as well as in the new searchable set; there may be overlap between searchable sets of two neighboring routers, depending on r. We define the *reachable set* of a router denoted by $\mathcal{S}_j^r$ as the set of all routers in the searchable sets of routers in $\mathcal{R}_{j,CP}$. If no router in $\mathcal{S}_j^r$ has f_i, it is downloaded from the CP and routed to the user following the backward path.

7.3 Abstraction of Cooperation

Cooperation Policy Design

Performance of a cooperation policy is determined by contents in the searchable set which is a function of r. The diversity of contents cached in this set increases caching efficiency which then calls for a caching scheme avoiding duplicate copies in the set [75, 134]. However, popular content may better be cached in multiple routers to be more accessible from all network edge routers. We model this tradeoff with parameter K which is the maximum number of content replicas in the network. In reality every file would have its own maximum number of copies which emerges automatically if r is fixed; we use a fixed K to illustrate system behavior across the whole parameter range. Using these two parameters, we name a cooperation policy with parameters K and r as *(K, r)-Cooperation Policy*, classified into four as follows:

1. Type I, **small** r**, small** K: Weak cooperation due to limited access to other caches and limited availability of popular content; the system is not using all its resources.

2. Type II, **small** r**, large** K: This is en-route caching. The most popular content is pushed to the network edge.

3. Type III, **large** r**, small** K: Network storage is effectively a single cache. Popular content is in network core.

4. Type IV, **large** r**, large** K: Strong cooperation. BHR and FPR cannot be improved at the same time since caching system is fully-utilized and reaches its Pareto frontier.

The complexity of cooperation can be calculated via communication and computation overhead [134]. Initially, all routers exchange their set of stored contents with routers in their searchable set. Assuming that each content is unit size and dropping the router index, this initialization step requires $O(MC|\mathcal{S}^c|)$ messages and results in $O(M^2C)$ message exchanges in the worst case. Upon a change in the cache of a router, this router informs all its r-hop neighbours about the evicted and admitted items. This per change announcement requires $O(|\mathcal{S}^c|)$ message in the worst case. In terms of computation, the cooperation does not involve any processing rather than discovering which of the replicas is closest to a specific router. Therefore, computation overhead is $O(|\mathcal{S}^c|)$.

Optimal Caching under (K, r)-Cooperation Policy

Assume a centralized entity knowing the content distribution at time t, $\mathbf{X}^t = [x_{i,j}]$ where $x_{i,j}$ is 1 if f_i is stored at R_j, and zero otherwise. At time t a user issues a request to R_l for f_u which is stored at R_{hit}. Denote the set of intermediate routers on the path between R_l and R_{hit} by $\mathcal{S}$, and the extended set $\mathcal{S} \cup R_{M+1}$ by $\mathcal{S}^+$. An optimal caching strategy (C_{OPT}) determines whether to cache f_u in the routers in $\mathcal{S}$, and if to be cached which items to evict in case of full cache occupancy.

A requested item can be served from edge router R_j or retrieved from another router R_k including the CP. Let $c_{j,k}$ denote the cost of downloading one byte at R_j from R_k. The cost function reflects the distance between the two entities and can be calculated using shortest path algorithms. For a (K, r)-Cooperation Policy, as the routers not in $\mathcal{S}_j^r$ are not reachable from R_j, we set $c_{j,k} = \infty$ if $R_k \notin \mathcal{S}_j^r$. Let our decision variables $\mathbf{Y}^t = [y_{i,j,k}] \in \{0, 1\}$ be 1 only if R_j downloads f_i from R_k. Note that if $y_{i,j,j} = 1$, then f_i is stored in R_j. C_{OPT} aims to minimize the total cost of serving all user requests arriving to all edge routers in the long run (first term in (7.1)) and cost of serving f_u for which user request is just received and can be cached in one of the routers in $\mathcal{S}$ (second term). C_{OPT} exploits its knowledge of current content distribution $\mathbf{X}^t$, file popularities ($\mathbf{p}$), and file size (s_i)

information as follows:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{L} \sum_{k=1}^{M+1} s_i p_i c_{j,k} y_{i,j,k} + s_u p_u \sum_{j=1}^{L} \sum_{R_k \in \mathcal{S}^+} c_{j,k} y_{u,j,k} \tag{7.1}$$

subject to (7.2)

Cache capacity constraints:

$$\sum_{i=1}^{N} s_i x_{i,j} y_{i,j,j} + s_u y_{u,j,j} (1 - x_{u,j}) \leq C_j \quad \forall R_j \in \mathcal{S} \tag{7.3}$$

$$\sum_{i=1}^{N} s_i y_{i,j,j} \leq C_j \quad \forall R_j \notin \mathcal{S} \tag{7.4}$$

Maximum replica constraint: $$\sum_{j=1}^{M} y_{i,j,j} \leq K \quad \forall i \tag{7.5}$$

Feasibility constraints:

$$y_{i,j,k} \leq y_{i,k,k} \qquad \forall i, \forall j, \forall k \tag{7.6}$$

$$y_{i,j,j} = x_{i,j} \quad \forall i, \forall R_j \notin \mathcal{S} \tag{7.7}$$

Service constraint: $$1 \leq \sum_{k=1}^{M+1} y_{i,j,k} \quad \forall i, \forall j \tag{7.8}$$

Availability constraint: $$y_{i,M+1,M+1} = 1 \quad \forall i. \tag{7.9}$$

Our objective (7.1) aims to minimize the serving cost by favoring the most popular files. *Cache capacity constraints* in (7.3) and (7.4) ensure the total size of items to be stored in a router's cache cannot exceed cache capacity. Only routers in $\mathcal{S}$ can consider putting the requested item f_u into their caches. *Maximum replica constraint* in (7.5) ensures that an item can have maximum K replicas in the network. Note that by removing this constraint, system can figure out optimal K for each neighborhood automatically. *Feasibility constraint* in (7.6) reflects f_i being retrievable from R_k only if R_k stores f_i whereas (7.7) states that contents cached by routers not in $\mathcal{S}$ do not change. *Service constraint* (7.8) forces the requests received at edge routers to be served from some location (i.e., local cache, another router's cache, or the CP), while *availability constraint* in (7.9) ensures all items are available from the CP. After C_{OPT} determines $\mathbf{Y}^t$, we update the new content distribution as $\mathbf{X}^{t+1} = \mathbf{Y}^t$ for the next decision instant. C_{OPT} is an integer programming problem which can be solved with optimization software for small instances of the problem but requires low-complexity distributed schemes for large scale networks. We

leave distributed solutions for future work.

Let $\mathcal{F}_j = \{u_{j,1}, u_{j,2}, u_{j,3}...\}$ be the list of user requests arriving at edge router R_j where $u_{j,i}$ is the i^{th} request for a file with size $s_{u_{j,i}}$. R_j can retrieve it only from its reachable set $\mathcal{S}_j^r$ which is defined as follow:

$$\mathcal{S}_j^r = \bigcup_{R_k \in \mathcal{R}_{j,CP}} \mathcal{S}_k^c. \tag{7.10}$$

A request will be counted as hit if at least one of the routers in $\mathcal{S}_j^r$ stores it. More formally, we define hit function $\delta_{j,i}$ for request $u_{j,i}$ (assuming $u_{j,i}$ is a request for f_i) as follows:

$$\delta_{j,i} = \begin{cases} 1 & \text{if } \sum_{R_k \in \mathcal{S}_j^r} y_{i,k,k} \geq 1 \\ 0 & \text{o/w.} \end{cases}$$

Next, we calculate BHR as follows:

$$BHR = \frac{\sum_{j=1}^{L} \sum_{u_{j,i} \in \mathcal{F}_j} s_{u_{j,i}} \delta_{j,i}}{\sum_{j=1}^{L} \sum_{u_{j,i} \in \mathcal{F}_j} s_{u_{j,i}}}. \tag{7.11}$$

If request $u_{j,i}$ is served from a router that is $h_{j,i}$ hops away from the user, and the path from R_j to the CP is H_j hops long, we can compute the FPR as follows:

$$FPR = 1 - \frac{\sum_{j=1}^{L} \sum_{u_{j,i} \in \mathcal{F}_j} s_{u_{j,i}} h_{j,i}}{\sum_{j=1}^{L} H_j \sum_{u_{j,i} \in \mathcal{F}_j} s_{u_{j,i}}}. \tag{7.12}$$

7.4 Interplay on Pareto Frontier

Setup and Metrics

We performed numerical evaluation on realistic and synthetic topologies. Realistic topologies are from [84], and synthetic topologies are scale-free networks of 50 nodes. Each node can store 25 objects. We present results on synthetic networks; realistic topologies produce similar results. Content popularity is modeled according to [83], and content set contains 5000 objects. We calculate the betweenness centrality (C_B) of each router in order to analyze its impact on cached content in a specific router under various (K, r) pairs. We define *coupling factor* (CPF) as the Pearson correlation between C_B and average popularity per bit in a node's cache; it measures topological impact on system performance. The rationale is that optimal

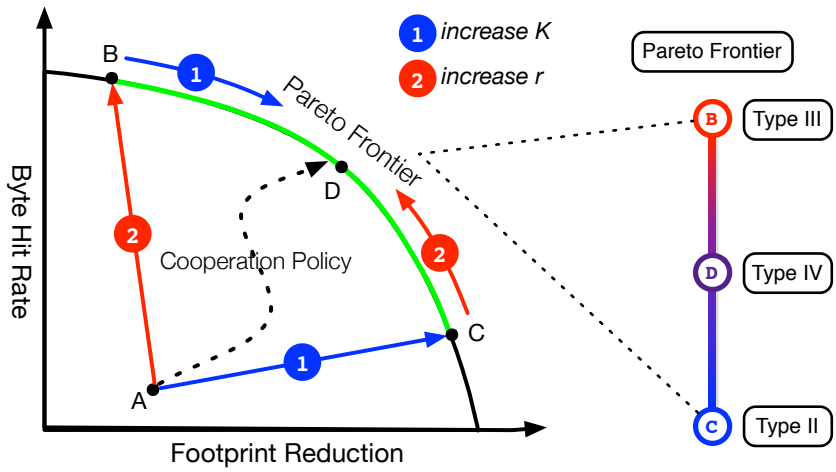


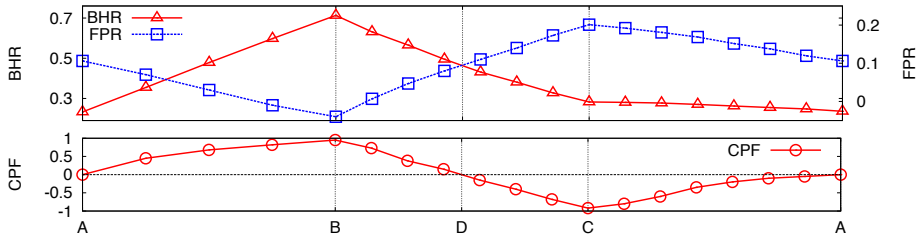
Figure 7.1: Conflicting BHR and FPR at the Pareto frontier. Type of cooperation at different points shown on right.

system performance is achieved by placing content at specific locations in a network according to its popularity and that C_B is a good metric to characterize a node's position in a graph. Strong correlation between the two indicates that content is tightly "coupled" with topology and topological properties influence system performance.

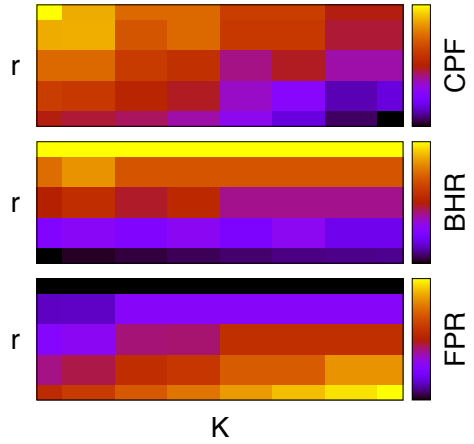
In the simulations, 30% of the edge routers are randomly selected and connected with client, and the server randomly connects to one of the 5 core nodes with highest C_B. Experiments were repeated at least 50 times.

Pareto Frontier

Fig. 7.1 shows how K and r impact caching performance. The solution to C_{OPT} provides the optimal cache profiles for given K and r (e.g., point A in Fig. 7.1), but it does not indicate the best values for these two parameters, i.e., we can improve performance by tuning K and r, because the system may be underutilized. However, our optimization model can be used to find Pareto frontier of the performance (green arc BC in Fig. 7.1). When we reach the Pareto frontier, we cannot improve BHR or FPR without hurting the other. The fan-shaped area defined by ABC is the area which a cooperation policy can explore to find the best tradeoff between K and r. Point D where we eventually reach the Pareto frontier depends on how cooperation policy balances BHR and FPR. Lines AB and AC are not parallel to the x- and y-axis, since changing either of r or K affects both BHR and FPR, as we show below.



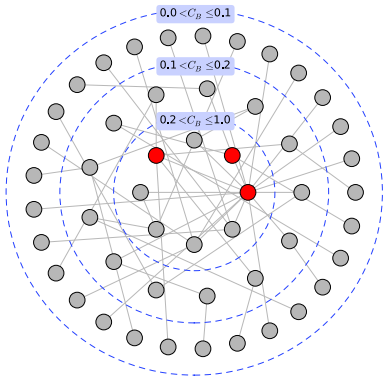
(a) Change in BHR (Top plot, left y-axis), FPR (top, right y-axis), and CPF (bottom) along AB, BC, and CA.



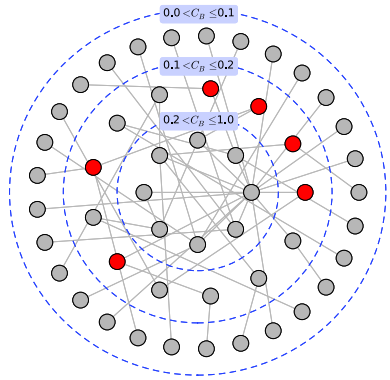
(b) CPF, BHR, and FPR as a function of K and r.

Figure 7.2: Performance of (K, r)-Cooperation along the boundary defined by ABC (a), and for all (K, r) pairs (b).

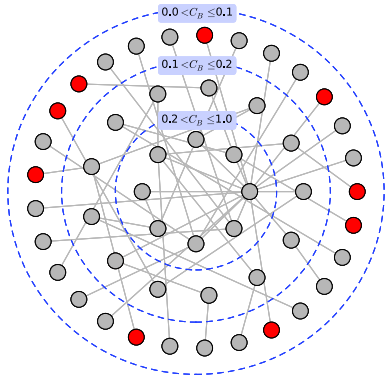
The upper graph in Fig. 7.2a shows how BHR and FPR vary as we move along the segments AB, BC, and CA, by varying r and K. Starting from A and moving clockwise (left to right in the figure), we increase the search radius which improves BHR, but decreases FPR due to additional search traffic or letting content be cached at routers with higher $h_{j,i}$ in (7.12). From B to C, along the Pareto frontier, we observe the tradeoff between BHR and FPR, with FPR reaching its maximum at C. From C to A, r is 0 so the system reduces to en-route caching where larger number of copies (near C) is beneficial, hence as we move towards A, both BHR and FPR decrease. Fig. 7.2b shows heatmaps of CPF, BHR, and FPR as function of K and r. Lighter values indicate higher values. It shows how BHR and FPR conflict each other, i.e., one achieving the highest performance while the other has the worst, in regions corresponding to the Pareto frontier.



(a) Point B in Fig. 7.1.



(b) Point D in Fig. 7.1.



(c) Point C in Fig. 7.1.

Figure 7.3: Content placement at points B, D, and C. Red color (dark dots) marks the most popular content. Nodes are grouped in circles according to C_B. Content migrates from core to edge as we move from B to C.

Coupling Content and Topology

The lower plot in Fig. 7.2a shows how CPF evolves along the same path. Values close to -1 or 1 indicate strong dependence between popularity and betweenness. A router with a high C_B is in or close to the core of the network whereas a router with low C_B is close to the network edge. At B, where CPF is close to 1, popular content is in nodes with high C_B, i.e., the core, whereas at C, where CPF is close to -1, it is at the edge where C_B is low. Along the Pareto frontier BC, we observe a "migration" of content from core to edge. At D where CPF is 0, both BHR and FPR are close to halfway point between their respective minima and maxima at B and C. We have observed this phenomenon across a wide range of experimental settings, but its full investigation is left for further study.

Fig. 7.3 shows cooperation policy's impact on content placement along the Pareto frontier BC. Point B (Fig. 7.3a), representing Type III cooperation favors BHR and places content in the core. Point D along BC (Fig. 7.3b) strikes a tradeoff between BHR and FPR and the content is neither in the core nor on the edge; this is Type IV cooperation. Finally, point C (Fig. 7.3c) favors FPR and pushes popular content to the edge.

Implications

Our results have profound implications on relationship between cooperation policies, content popularity, and network topology. They also give us hints on when and how topological properties should be taken into account in caching strategy design. We summarize the main implications as follows:

1. Cooperation pushes performance to Pareto frontier and couples content popularity and topological properties together. How and where it falls on the frontier depends on how it balances BHR and FPR.

2. Content popularity and topology strongly correlate with each other only close to the Pareto frontier. Whether the correlation is positive or negative depends on how the cooperation policy favors one of the two metrics.

3. The optimization model implies that C_B has more influence on performance when we get closer to points A or B in Fig. 7.1; only Type II and III cooperation policies can fully utilize C_B to enhance performance.

4. We conjecture that tight coupling between content popularity and topology comes similar mathematical structures as both exhibit power-law properties. Were popularity closer to uniform or topology closer to a random network, this tight coupling might disappear.

7.5 Conclusion

We modeled cache cooperation by its search radius and tolerance of duplicates. We performed a thorough numerical analysis and showed that cooperation policy pushes system performance to its Pareto frontier, and how it couples content with topology. We proposed a way to measure impact of topology on system performance. We show when and how topological information should be taken into account in in-network caching strategy design.

Chapter 8

Optimal Chunking and Partial Caching

Traditionally caches store complete objects, but video files and the recent emergence of information-centric networking have highlighted a need for understanding how partial caching could be beneficial. In partial caching, objects are divided into chunks which are cached either independently or by exploiting common properties of chunks of the same file. In this chapter, we identify why partial caching is beneficial, and propose a way to quantify the benefit. We develop an optimal n-Chunking algorithm with complexity $O(ns^2)$ for an s-byte file, and compare it with ϵ-optimal homogeneous chunking, where ϵ is bounded by $\mathcal{O}(n^{-2})$. Our analytical results and comparison lead to the surprising conclusion that neither sophisticated partial caching algorithm nor high complexity optimal chunking are needed in information-centric networks. Instead, simple utility-based in-network caching algorithm and low complexity homogeneous chunking are sufficient to achieve the most benefits of partial caching.

8.1 Introduction

Network caching reduces network traffic by exploiting redundancy in traffic [112] and popularity of content [31]. Different caching strategies have been proposed for different cases, such as web and media caching. An important, yet not widely studied question in caching relates to whether objects should be cached in their entirety (*integral caching*) or if only parts of objects should be cached (*partial caching*). Traditionally, web caching has favored the integral approach, whereas media caching has also considered partial caching.

Information-centric networking (ICN) [1–3] uses caching extensively for content delivery and some ICN approaches by default divide objects into chunks, leading effectively to partial caching. This is suitable for large video files, since they are often only partially accessed [34]. Integral caching may waste cache space by storing parts of objects that are less popular than the most popular parts of the same objects. It appears intuitive to develop efficient partial caching algorithms for optimal handling of such differing popularities inside objects.

However, the reality is somewhat more nuanced, as we show in this chapter. Our focus is on understanding how different ways of dividing the object (*chunking*) reflect on performance of caching and how to optimally chunk an object. We show that, while partial caching is useful for partially accessed objects, similar results can be achieved via chunking the object into enough many chunks and using simple caching algorithms. In other words, there is only a very small range of system parameters where sophisticated partial caching algorithms are needed, even with erratic object access patterns.

Specifically, the contributions of the chapter are as follows:

- We analyze the effects of chunking and develop the concept of *popularity distribution distance* to measure the effectiveness of a chunking scheme. We derive bounds on performance of partial caching and compare the optimal chunking with naive homogeneous chunking analytically.

- We demonstrate the performance of partial caching algorithms with different chunking schemes and the experiments confirm our analysis that after a moderate number of chunks, partial caching yields no further benefits.

- Both analytical and experimental results show neither sophisticated partial caching algorithm nor optimal chunking are needed in practice. Instead, a simple utility-based caching algorithm with naive homogeneous chunking is sufficient to achieve most benefits of partial caching.

The rest of the chapter is organized as follows. Section 8.2 describes our system model and Section 8.3 presents formal analysis on chunking schemes. Section 8.4 formalizes optimal caching algorithms and evaluates their performance under different chunking schemes. Section 8.5 introduces a utility-based heuristic and compares its performance with the optimal caching algorithms. Section 5.4 reviews related work and Section 9.9 concludes the chapter.

Table 8.1: Summary of notations.

Notation	Meaning
f_i, $f_{i,j}$	File i and chunk j of file i
n_i	Number of chunks in file i
p_i, $p_{i,j}$	Popularity of file i and popularity of chunk j of file i
s_i, $s_{i,j}$	Size of file i and size of chunk j of file i
N	Number of files
M	Number of routers
L	Number of routers that are directly connected with users
R_i	Router i
C_i	Storage capacity of router i
CP	Content Provider
R_{hit}	Router that holds the requested content
$\mathcal{S}$	Set of routers between an edge router and R_{hit}
$\mathbf{X}^t$	Content distribution on routers at time t
$X_{i,j,k,k}$	Decision variable for storing $f_{i,j}$ at R_k
$X_{i,j,k,k'}$	Decision variable for retrieving $f_{i,j}$ from $R_{k'}$ by R_k
$c_{k,k'}$	Cost of retrieving one byte from $R_{k'}$ to R_k
$\mathbf{p}, \tilde{\mathbf{p}}$	Real and observed popularity vector
$\mathcal{M}, \tilde{\mathcal{M}}$	Real and observed user request patterns

8.2 System Model

Consider a network of M routers organized in a general topology. Let R_i denote the router i with cache storage capacity of C_i bytes. This network serves the users that generate requests for files in the set $\mathcal{I}$ with $|\mathcal{I}| = N$. We denote a file by f_i and its size by s_i. All files are stored permanently at the Content Provider (CP) which is represented as the $(M+1)^{\text{th}}$ router (R_{M+1}). Users interact only with the L *edge routers* – routers connect to the users – also referred to as leaf nodes[1]. A file f_i is divided into n_i smaller units referred to as *chunks*, and j^{th} chunk is denoted as $f_{i,j}$. Denote the probability of request for a file by this file's *popularity* p_i, and similarly denote the popularity of chunk $f_{i,j}$ by $p_{i,j}$. We refer to the popularity vector in both cases by $\mathbf{p} = [p_i]$ (or $\mathbf{p} = [p_{i,j}]$). If an edge router has the requested item in its cache, we call this a *hit* and this item is transmitted to the user directly from this router. In case of a *miss* – the case where the router does not have the item, the request is retrieved from the closest router storing this item. If the item is not stored in the network, it is retrieved from CP.

8.3 Analysis on Content Chunking

Cutting a file into smaller *chunks* improves caching performance since more fine-grained caching decisions can be made, especially when different parts of the file have different popularities. However, quantifying the effects of chunking and the resulting benefits in partial caching have largely been unexplored. We now present the relationship between chunking and performance, then quantify the benefits of partial caching, and outline the steps of an optimal chunking algorithm.

Chunking Effect: Origin of Partial Caching Benefit

Assume that the smallest indivisible unit of a file is one byte and the smallest unit requested by the user is a chunk. Fig. 8.1 gives an example where a six-byte file is divided into two chunks A and B. Users can access individual bytes in an arbitrary manner and we denote this "real user access pattern" as $\mathcal{M}$. Because of chunking, the real access pattern has to be translated into coarser granularity chunk access pattern, from $\mathcal{M}$ to $\tilde{\mathcal{M}}$ as in Fig. 8.1. This distorts the popularity distribution of the bytes since unpopular bytes could be in the same chunk as popular bytes (e.g., bytes 1 and 2 in the figure), thus inflating their observed popularity. Due to this distortion,

[1] We use node, router, and cache interchangeably.

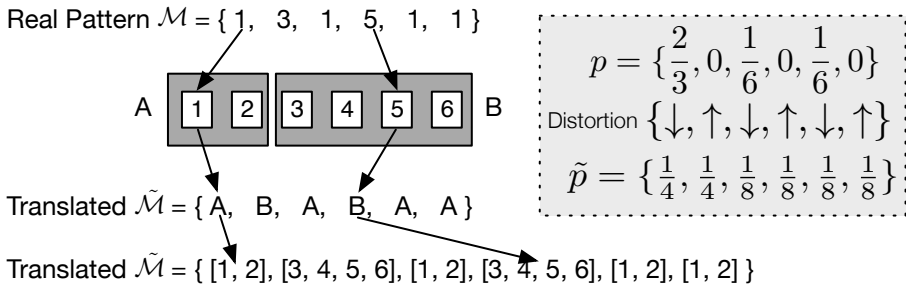


Figure 8.1: Illustration of chunking effect for a chunking scheme with two chunks A and B.

although the original popularity distribution is $\mathbf{p} = \{\frac{2}{3}, 0, \frac{1}{6}, 0, \frac{1}{6}, 0\}$, the translated popularity distribution becomes $\tilde{\mathbf{p}} = \{\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}\}$. We call this distortion from the real $\mathbf{p}$ to the translated $\tilde{\mathbf{p}}$ *chunking effect*.

Chunking effect leads to the popularity of the bytes in the same chunk to be equal which often also means over- or under-estimation of popularity. Popularity estimate has direct impact on caching performance:

1. Overestimated popularity increases the chance of caching unpopular content.

2. Underestimated popularity decreases the chance of caching popular content.

Both cases lead to a failure to use cache efficiently because of wasting cache space on unpopular content.

The effect can happen anywhere in the network: at the client, at the server, or at a network cache in a content router. Nonetheless, the effect is the same since the bytes in the same chunk will be given the same popularity and we lose the information from the real sequence. Vanichpun et al. [135] show that for most demand-driven caching algorithms (e.g., LRU, LFU), it is reasonable to assume that the closer $\tilde{\mathbf{p}}$ is to $\mathbf{p}$, the better caching decision a caching algorithm can make, therefore achieve higher caching performance.

Popularity Distribution Distance: Quantifying the Benefits

Multiple metrics could be used to measure information loss due to the chunking effect. However, simple difference of two distributions has very direct connection to the performance.

Vanichpun et al. [135] show that probability of an object being cached is a function of its translated probability. Let $\mathcal{F}(\tilde{\mathcal{M}}, C) \rightarrow \mathbf{x}$ be a caching algorithm which maps a translated request pattern $\tilde{\mathcal{M}}$ and cache capacity C to a caching decision vector $\mathbf{x} = \{x_1, x_2, x_3...\}$, where x_i specifies the probability that item f_i should be kept in the cache. Let $\mathcal{I}$ denote the set of the whole content items with N elements and $\mathcal{I}_{\mathcal{F}}$ the items cached by $\mathcal{F}$. Assuming unit size items, the total size of the items in the cache sums to the cache capacity: $\sum_{i=1}^{N} x_i = C$. Let us introduce $\mathbf{v} = \{\frac{x_1}{C}, \ldots, \frac{x_i}{C}, \ldots, \frac{x_N}{C}\}$, which is simply normalized version of $\mathbf{x}$.

The performance of $\mathcal{F}$ can be evaluated by its (byte) hit rate $\mathcal{H}$ which is simply the joint probability of an incoming request being for a specific content f_i and this item f_i being stored in the cache. We calculate $\mathcal{H}$ as follows:

$$\mathcal{H} = P(f \in \mathcal{I}_{\mathcal{F}}) = \sum_{i=1}^{|\mathcal{I}|} P(f_i, f_i \in \mathcal{I}_{\mathcal{F}}). \tag{8.1}$$

Since the event that next request is f_i is independent from the event that f_i is in the cache, Eq. (8.1) can be rewritten as:

$$\mathcal{H} = \sum_{i=1}^{|\mathcal{I}|} P(f_i) P(f_i \in \mathcal{I}_{\mathcal{F}}) = C\mathbf{p}\mathbf{v}^T. \tag{8.2}$$

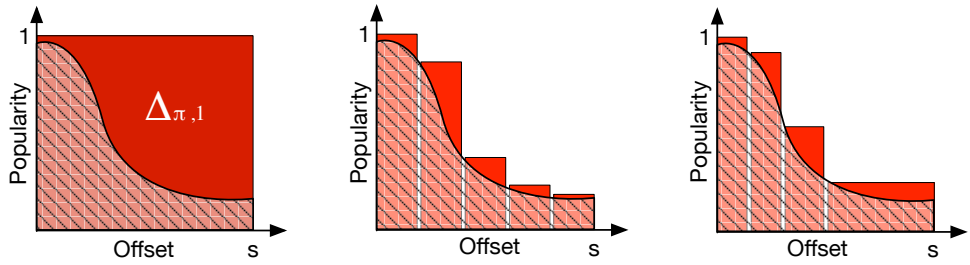
From Eq. (8.2), hit rate can be viewed as a function of cache size and the dot product of two distributions. Let $\mathcal{X}_k$ denote the set of k most popular objects, then the optimal caching decision is:

$$\mathbf{v}^* = \begin{cases} \frac{1}{C} & \text{if } v \in \mathcal{X}_C \\ 0 & \text{if } v \notin \mathcal{X}_C. \end{cases}$$

Caching decision $\mathbf{v}$ is completely determined by a specific caching algorithm. How close and how fast $\mathbf{v}$ converges to $\mathbf{v}^*$ is an important metric to measure the quality of a caching algorithm. Let $\mathcal{H}^*$ be the hit rate if the caching decision is optimal ($\mathbf{v}^*$). Then, the performance gap between a non-optimal and the optimal scheme is calculated as follows:

$$\Delta\mathcal{H} = \mathcal{H}^* - \tilde{\mathcal{H}} = C\mathbf{p}(\mathbf{v}^* - \mathbf{v})^T. \tag{8.3}$$

Similarly, Eq. (8.3) can also be used to compare the performance difference of two caching algorithms or two chunking schemes under a specific caching algorithm. However, in most cases, $\mathcal{F}(\tilde{\mathcal{M}}, C) \rightarrow \mathbf{v}$ is too complicated to provide any useful information. To get around this, we measure the "expected performance loss" instead. As argued above, the less information we lose in the request series, the better caching decision we can



(a) Integral caching. (b) Homogeneous chunks. (c) Heterogeneous chunks.

Figure 8.2: Benefit of chunking.

make. Then, given the distribution $\tilde{\mathbf{p}}$ which is expected by a caching algorithm, what would be the performance loss if the actual content popularity based on user request is $\mathbf{p}$? Expected performance loss can be calculated as follows:

$$\Delta\mathcal{H} = \mathcal{H} - \tilde{\mathcal{H}} = C(\mathbf{p} - \tilde{\mathbf{p}})\mathbf{v}^T \propto (\mathbf{p} - \tilde{\mathbf{p}}). \tag{8.4}$$

In a sense, the "expected loss" shows the potential benefit from partial caching. If we define $|\mathbf{p} - \tilde{\mathbf{p}}|$ as the *popularity distribution distance* to measure the difference of two distributions, Eq. (8.4) shows that the potential performance gain of a partial caching algorithm positively correlates to the popularity distribution distance.

Performance Bound: Decaying Speed of the Benefit

Let $\pi_i(y)$ denote the probability of accessing the byte at position y of file f_i. In other words, it is the popularity of y^{th} byte in f_i. We assume that $\pi_i(y)$ is a continuous function and has at least second derivative over $[0, s_i]$. We further denote $\Pi_i(y)$ as the popularity observed from a specific chunking. Since all bytes in a chunk have the same popularity, $\Pi_i(y)$ is a step function. While $\pi_i(y)$ is the real popularity distribution (as the discrete $\mathbf{p}$), $\Pi_i(y)$ is the perceived one as $\tilde{\mathbf{p}}$.

Fig. 8.2 illustrates why we can benefit from chunking. x-axis is the offset in a file, and y-axis is the conditional probability of this offset byte being accessed, i.e., the popularity of the specific byte located at this offset. The curved boundary of the shaded area in the figure represents the real popularity $\pi_i(y)$. The solid red bars represent the popularity assigned for

each chunk in the file. *Integral caching* can be considered as a special case of partial caching with only one chunk and $\Pi_i(y) = 1$ as Fig. 8.2a shows. The area above the curve $\pi_i(y)$ is the overestimation part. Naturally, chunking a file into smaller pieces reduces the difference of the two areas.

From Eq. (8.4), we can interpret that quantifying the benefit of chunking is equivalent to measuring the distance of two models derived from $\pi_i(y)$ and $\Pi_i(y)$ respectively. We calculate the chunking benefit as follows:

$$\Delta_{\pi_i,\Pi_i} = \frac{1}{s_i}\int_0^{s_i} \big|\pi_i(y) - \Pi_i(y)\big| dy. \tag{8.5}$$

Δ_{π_i,Π_i} is normalized over the file size so that we can compare files of different sizes. Based on the definition of Δ_{π_i,Π_i}, we can further define the upper bound of benefit from partial caching to integral caching as $\Delta_{\pi_i,1}$, where 1 stands for uniform chunk popularity function $\Pi_i(y) = 1$ for all $y \leqslant s_i$. Obviously, the smaller the Δ_{π_i,Π_i} is, the more accurately $\Pi_i(y)$ describes user's real access pattern. Then, improving chunking performance boils down to chunking a file in a way that minimizes the difference Δ_{π_i,Π_i}. A follow-up question is how fast it converges to the actual access pattern, i.e., how quickly Δ_{π_i,Π_i} converges to 0.

Because $\pi_i(y)$ may be arbitrary distribution, deriving the exact decaying speed is difficult. However, $\Pi_i(y)$ is a step function, and from Eq. (8.5), we can see the integration on the second term is actually the Riemann sum of $\pi_i(y)$ over $[0, s_i]$, as the area of rectangles shows in Fig. 8.2. Suppose we use midpoint Riemann sum for obtaining a tighter bound, and let the middle point represented by $\bar{y}_k = \frac{1}{2}(y_{k-1} + y_k)$. Then $\Pi_i(y)$ can be defined as follows:

$$\Pi_i(y) = \pi_i(\frac{y_{k-1} + y_k}{2}) = \pi_i(\bar{y}_k) \quad \forall y \in [y_{k-1}, y_k).$$

Using Taylor series expansion on $\pi_i(y)$, we calculate the bound of the popularity distribution distance for the k^{th} chunk as follows:

$$\begin{aligned}
\big|\delta_{i,k}\big| &= \left|\int_{y_{k-1}}^{y_k} \big|\pi_i(y) - \Pi_i(y)\big| dy\right| \\
&= \int_{y_{k-1}}^{y_k} \big|\pi_i'(\bar{y}_k)(y - \bar{y}_k) + \frac{1}{2}\pi_i''(\bar{y}_k)(y - \bar{y}_k)^2\big| dy \\
&\leq \frac{1}{24}(y_k - y_{k-1})^3 \max_{[0,s_i]} |\pi_i''(y)|.
\end{aligned}$$

Summing over n chunks, we can get the bound for overall popularity distribution distance as follows:

$$\begin{aligned}\left|\Delta_{\pi_i,\Pi_i}\right| &= \left|\frac{1}{s_i}\int_0^{s_i}\left|\pi_i(y)-\Pi_i(y)\right|dy\right| \\ &\leq \frac{1}{s_i}\times[\frac{1}{24}\times\frac{(s_i-0)^3}{n^2}\times\max_{[0,s_i]}|\pi_i''(y)|] \\ &= \frac{\max_{[0,s_i]}|\pi_i''(y)|}{24}\times\frac{s_i^2}{n^2}.\end{aligned}$$

The second term is the square of the average chunk size. If we consider the chunk popularity and file size as given, we can replace the first term with constant c. s_i/n is the average chunk size, denoted as $\bar{s}$. The formula[2] can be rewritten as:

$$\left|\Delta_{\pi_i,\Pi_i}\right| \leq c\times(\frac{s_i}{n})^2 = \mathcal{O}(\bar{s}^2) = \mathcal{O}(n^{-2}). \tag{8.6}$$

This shows that the popularity distribution distance is bounded by the square of the average chunk size. For a given file, the convergence speed of model distance is inversely proportional to n^2. In practical terms, this implies that there is a specific number of chunks for every file after which additional chunks bring only negligible benefits in performance.

Conclusion:

1. *Performance of partial caching strategy positively correlates to the popularity distribution distance, which decays with the speed bounded by $\mathcal{O}(\bar{s}^2)$ and $\mathcal{O}(n^{-2})$.*

2. *Given a target caching performance, the number of chunks needed increases linearly as file size grows.*

Optimal Chunking and Complexity Analysis

Obviously, smaller chunk size, $\bar{s}$, reduces popularity distribution distance Δ_{π_i,Π_i} and results in Δ_{π_i,Π_i} asymptotically approaching 0. On one hand, this improves caching performance, but on the other hand, too many chunks increases maintenance overheads and may even degrade router performance in some cases. Moreover, the minimum chunk size may also be restricted by hardware, minimum video clip size, and other practical limits. Hence, we want to use as few chunks as possible to achieve sufficiently small Δ_{π_i,Π_i}.

[2]If Riemann sum is calculated using alternate methods, e.g., left, right, minimum or maximum sum, the bound changes to $\mathcal{O}(\bar{s})$ and $\mathcal{O}(n^{-1})$.

We now present an optimal chunking algorithm and analyze its complexity. First, for a given number of chunks n, we divide a file into chunks to achieve the minimum Δ_{π_i,Π_i}. Next, we find the smallest number of chunks to attain a target Δ_{π_i,Π_i}.

Question 1: Given that an s-byte file is to be divided into n chunks where $s \gg n$, how can we find the optimal chunking that achieves the minimum popularity distribution distance $\Delta_{\min}$? Suppose the smallest unit of operation is one byte and the popularity of each byte is known. A naive algorithm goes through all the possible partitions and selects the one achieving $\Delta_{\min}$. Since there are $\binom{s-1}{n-1}$ ways of dividing an s-byte file into n chunks, the computational complexity equals to $\binom{s-1}{n-1} = \frac{(s-1)!}{(n-1)!(s-n)!} = \frac{1}{(n-1)!}s^{n-1} + o(s^{n-1})$ showing the time complexity of naive algorithm to be $\mathcal{O}(s^{n-1})$, and space complexity to be $\mathcal{O}(n)$.

This optimization problem exhibits obvious "optimal substructure". Assume we look for a solution which produces $n = n_1 + n_2$ chunks, and the first n_1 chunks cover $[0, i]$ bytes, and the remaining n_2 chunks cover $[i+1, s-1]$ bytes. We can first focus on $[0, i]$ and optimize it, then turn to optimize $[i+1, s-1]$. By combining the optimal solutions of both parts, we have the optimal partition of the whole file. Let $\Delta|^n_{[a,b]}$ be the minimum popularity distribution distance for the file's bytes between $[a, b]$ in case of n chunks. The minimum distance can be recursively found as follows:

$$\Delta_{\min} = \Delta|^n_{[0,s-1]} = \min_{0 \le i < s} (\Delta|^1_{[0,i]} + \Delta|^{n-1}_{[i+1,s-1]}). \tag{8.7}$$

The top-down recursive solution Eq. (8.7) also tries all the possible cutting points. The induced recursive tree has degree $\mathcal{O}(s)$, and depth $n-1$ (restricted by the number of chunks n). Although simple analysis shows that it suffers from the same time complexity $\mathcal{O}(s^{n-1})$ as the naive algorithm, its optimal substructure property helps us reducing the search complexity by eliminating redundant calculations.

Algorithm 7 shows our heterogeneous n-Chunking algorithm using bottom-up dynamic programming. The algorithm first constructs two tables X and X' of size $n \times s$. Element $X[i, j]$ stores the optimal Δ over bytes $[0, j]$ by cutting it into $i+1$ chunks. $X'[i, j]$ stores the starting point of the last chunk in an optimal chunking scheme over $[0, j]$ with $i+1$ chunks. Then the algorithm constructs another table Y of size s^2, where $Y[i, j]$ stores $\Delta|^1_{[i,j]}$. Filling $X[0, j]$, $X[i, 0]$, and $Y[i, j]$ are trivial. Lines (8 – 17) are for the general case; the three loops build the table row by row by filling each cell $X[i, j]$ with optimal $\Delta|^{i+1}_{[0,j]}$ with the given chunk number (i.e. $i+1$) and data range (i.e. $[0, j]$). There are $\mathcal{O}(n \times s)$ entries in X, and for each

1: **Input:** s-byte file, n chunks
2: **Output:** chunking scheme
3: Construct three tables X, X' and Y.
4: Set $X[i,j] = +\infty \quad \forall i \in [1, n-1], \forall j \in [1, s]$
5: Set $X[i,0] = \Delta|_{[0,0]}^{i+1} \quad \forall i \in [0, n-1]$
6: Set $X[0,j] = \Delta|_{[0,j]}^{1} \quad \forall j \in [0, s-1]$
7: Set $Y[i,j] = \Delta|_{[i,j]}^{1} \quad \forall i \leq j \in [0, s-1]$
8: **for** $i = 1 \rightarrow n-1$ **do**
9: **for** $j = 1 \rightarrow s-1$ **do**
10: **for** $j' = 1 \rightarrow j-1$ **do**
11: **if** $X[i,j] > X[i-1,j'] + Y[j'+1,j]$ **then**
12: $X[i,j] = X[i-1,j'] + Y[j'+1,j]$
13: $X'[i,j] = j'$
14: **end if**
15: **end for**
16: **end for**
17: **end for**
18: Return X, X'

Algorithm 7: Heterogeneous n-Chunking

entry, we need to consider $\mathcal{O}(s)$ cases. Therefore the time complexity of n-Chunking is $\mathcal{O}(ns^2)$, and space complexity is $\mathcal{O}(s^2)$. The optimal chunking scheme can be easily constructed from X' returned by n-Chunking and the corresponding $\Delta_{\min}$ is stored at $X[n-1, s-1]$.

Question 2: Given a target popularity distribution distance Δ, what is the minimum number of chunks $n_{\min}$ that achieves the desired Δ? This problem can be considered as an extension of the previous one, and can be solved by slightly adopting n-Chunking algorithm. We can replace the first deterministic n-round loop with a *while* loop, where we keep comparing Δ with $X[n', s-1]$ (n' is the current round), and break the loop when $\Delta > X[n', s-1]$. Then $n_{\min} = n'$ is the minimum number of chunks to achieve Δ, and X' contains the corresponding chunking scheme. Tables X and X' grow during the calculation. The complexity then depends on the number of iterations $n_{\min}$, and our analysis in Section 8.3 shows $n_{\min} \leq c^{\frac{1}{2}} s \left|\Delta\right|^{-\frac{1}{2}}$. $\mathcal{O}(n_{\min} s^2)$ and $\mathcal{O}(s^2)$ are time and space complexities for solving Question 2, respectively.

Homogeneous Chunking: Low-Complexity Alternative

Figure 8.2b and 8.2c show examples of homogeneous and heterogeneous chunking schemes. n-Chunking is an example of heterogeneous chunking and it requires file internal popularity distribution; its complexity is also considerably high. On the contrary, a homogeneous chunking does not require any information about popularity, as it simply divides the object into equal-sized chunks. It can be easily extrapolated from the conclusions in Section 8.3, that the performance difference of n-Chunking and a homogeneous chunking is bounded by $\mathcal{O}(\bar{s}^2)$ and $\mathcal{O}(n^{-2})$. In other words, as the number of chunks increases, the difference between the schemes diminishes.

Even though there are concerns that the increased number of chunks may bring up management overheads, [136, 137] actually showed the feasibility of caching many small chunks without degrading the performance in reality. Practically, it indicates there is no need to incorporate complicated optimal chunking scheme in the system. As our results show, after a moderate number of chunks, there is essentially no difference between the optimal and the homogeneous chunking.

Conclusion: *Homogeneous chunking is ϵ-optimal where $\epsilon = \frac{cs_i^2}{n^2}$. In other words, the difference between homogeneous chunking and n-Chunking is bounded by $\mathcal{O}(n^{-2})$.*

8.4 Analysis on Caching Systems

To verify our analysis in Section 8.3 and obtain performance bounds, we now evaluate different chunking schemes on a caching system. Please refer to Table 8.1 for the notation.

Caching System Model

Assume that there is a centralized entity aware of the network conditions. In particular, the content distribution at time t denoted by $\mathbf{X}^t = [x^t_{i,j,k}]$ is available to this entity. If chunk $f_{i,j}$ is stored at node R_k, then $x^t_{i,j,k}$ is 1 and zero otherwise. We devise a strategy called *dynamic partial caching* (P_{DyPa}) that gives a decision at each time instant when a user requests a chunk $f_{u,v}$, i.e., it dynamically adapts the cached contents. Suppose at time t chunk $f_{u,v}$ is stored at node R_{hit} and a user requests via a leaf node R_l. R_l will retrieve this item from R_{hit}, and each router on the path from R_{hit} to R_l must decide: cache this object or not. In case the item is cached in any of the nodes, some items stored in the cache may need to be evicted from the cache. We refer the set of all these intermediate nodes as $\mathcal{S}$.

An optimal caching strategy minimizes the cost of serving the whole user requests by storing the items at the most appropriate routers and by favoring the most popular chunks of the most popular files. Let $c_{k,k'}$ denote the cost function for fetching one byte from $R_{k'}$ to R_k. It reflects the distance between the two entities which can be calculated using shortest path algorithms, e.g., Dijkstra. The optimal strategy decides if chunk $f_{i,j}$ is to be stored at R_k and if not, from which router $R_{k'}$ to be fetched. Since this decision determines the new content distribution at time $t+1$, we define our decision variables as $\mathbf{X}^{t+1}$. Let our binary decision variable $x_{i,j,k}^{t+1}$ be 1 if chunk $f_{i,j}$ is to be stored at R_k. Similarly, let $x_{i,j,k,k'}^{t+1}$ be 1 if R_k downloads $f_{i,j}$ from $R_{k'}$. For harmony of notation, we re-define the content distribution by $\mathbf{X}^t = [x_{i,j,k,k}^t]$. Given $\mathbf{X}^t$, the number of chunks in a file i (n_i), file popularity (p_i), chunk popularity ($p_{i,j}$), and chunk size ($s_{i,j}$) information, using the approach provided in [138], we can formulate P_{DyPa} as follows:

$$P_{\text{DyPa}} : \min\Big(\sum_{k=1}^{L}\sum_{i=1}^{N}\sum_{j=1}^{n_i}\sum_{k'=1}^{M+1} s_{i,j}p_i p_{i,j} c_{k,k'} x_{i,j,k,k'}^{t+1} x_{i,j,k',k'}^{t+1} + s_{u,v}p_u p_{u,v}\Big(\sum_{k=1}^{L}\sum_{\forall R_{k'}\in\mathcal{S}\cup R_{M+1}} c_{k,k'} x_{u,v,k,k'}^{t+1}\Big)\Big) \tag{8.8}$$

subject to:

$$\sum_{i=1}^{N}\sum_{j=1}^{n_i} s_{i,j}x_{i,j,k,k}^{t}x_{i,j,k,k}^{t+1} + s_{u,v}x_{u,v,k,k}^{t+1}(1-x_{u,v,k,k}^{t}) \le C_k, \forall R_k \tag{8.9}$$

$$x_{i,j,k,k'}^{t+1} \le x_{i,j,k',k'}^{t+1} \qquad \forall i, \forall j, \forall k, \forall k' \tag{8.10}$$

$$1 \le \sum_{k'=1}^{M+1} x_{i,j,k,k'}^{t+1} \quad \forall i, \forall j, \forall k \tag{8.11}$$

$$x_{i,j,M+1,M+1}^{t+1} = 1 \quad \forall i, \forall j \tag{8.12}$$

$$x_{i,j,k,k'}^{t+1} \in \{0,1\} \qquad \forall i, \forall j, \forall k, \forall k'. \tag{8.13}$$

Our objective (8.8) includes the cost of serving the request from the edge router's cache as well as the cost of serving from any other content router in the ISP network and CP (i.e., R_{M+1}). The first term in (8.8) represents the cost of serving the requests for the contents already existing in the system whereas the second term is for serving the current request for item $f_{u,v}$. Please note that if $x_{i,j,k,k} = 1$, then $f_{i,j}$ is stored in R_k. Const. (8.9) ensures the total size of items to be stored in a cache cannot

exceed cache capacity. Const. (8.10) reflects the fact that $f_{i,j}$ can be fetched from $R_{k'}$ only if $R_{k'}$ stores $f_{i,j}$. Const. (8.11) forces the content item to be served from some location (i.e., local cache, another router's cache, or the CP) while Const. (8.12) states that all items are permanently stored in the CP. Const. (8.13) defines the type of variables. The problem in (8.8-8.13) is an integer linear programming problem (ILP) which can be solved by an optimization software once $c_{k,k'}$ values are computed.

If we assume that file and chunk popularities already contain sufficient information to describe user behavior, we can further simplify the optimal caching problem into a static content placement problem, i.e., the set of items to be stored at each cache is decided once and no changes are done. In this approach, content distribution is time-invariant leading to $\mathbf{X}^{t+1} = \mathbf{X}^t$ for allÂ t. Hence, we can simplify P_{DyPa} using this equality to derive P_{StPa} as follows:

$$P_{\text{StPa}} : \min \sum_{k=1}^{L} \sum_{i=1}^{N} \sum_{j=1}^{n_i} \sum_{k'=1}^{M+1} s_{i,j} p_i p_{i,j} c_{k,k'} x_{i,j,k,k'}. \tag{8.14}$$

The constraints would need to be similarly modified. Since integral caching is a special case of partial caching with $n_i = 1$ in Eq.(8.14), we skip the formulation for dynamic integral caching (DyIn).

Compared with the dynamic model, the static model does not take current content distribution into account and is unaware of the specific user request sequence. The static model aims to achieve the long-term optimality in content placement while the dynamic model attempts to find the best next caching decision based on the current content distribution in the network. Besides both problems being hard to solve as ILP problems [139], they require centralized global knowledge which is infeasible in real networks. We use these models as benchmarks for our lower-complexity heuristic in Section 8.5.

Metrics and Setup

- **Byte hit rate (BHR):** Byte hit rate is the percent of requested data that can be served by the cache within the network. It measures savings in outgoing traffic.

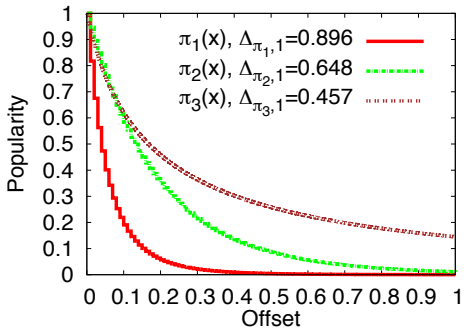
- **Footprint reduction (FPR):** Footprint reduction is the product of traffic volume and the distance it travels in the network. It measures traffic reduction inside the network compared to retrieving the content from the CP.

- **Cache-level similarity** (γ_c) measures how much the same content is stored at the caches of two caching strategies subject to analysis. Hence, cache-level similarity measures the average similarity over all routers in the network under two caching strategies. We use *Jaccard similarity* [140] for evaluating the similarity of two routers.
- **Network-level similarity** (γ_n) considers the whole caches as a single huge cache and compares the similarity of the caches of the two networks under comparison. Network-level similarity calculated using Jaccard similarity ignores the exact storage location and focuses on if an item is stored in the network or not.

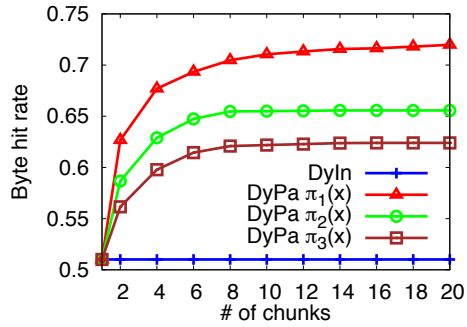
Cheng et al. [141] show that Youtube videos' popularity follows Weibull distribution with shape parameter $k = 0.513$; we use this setting in the evaluation. We also use Weibull distribution to model chunk popularity with different shape parameters. Because there is no evidence showing that file size correlates with its popularity, we select file sizes from $(5, 15)$ MB uniformly. Each video is chunked with the heterogeneous n-Chunking algorithm and user request pattern follows Independent Reference Model (IRM). Due to the space limitations, we only present the results on the 4-level 2-tree topology with model parameters set to $M = 5$, $N = 100$ and $C = 50$ MB. We verified various parameter settings on different topologies and results agree with those shown here.

Partial Caching vs. Integral Caching

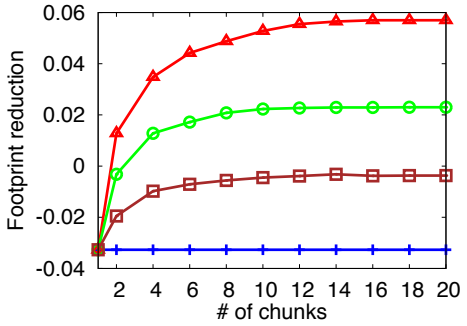
Fig. 8.3a shows three Weibull distributions with different shape parameters (0.4, 0.6 and 0.8 for π_1, π_2 and π_3 respectively) to model the internal popularity. The partial caching benefit for these distributions are $\Delta_{\pi_1,1} = 0.896$, $\Delta_{\pi_2,1} = 0.648$ and $\Delta_{\pi_3,1} = 0.457$. The calculation implies distribution π_1 has the largest benefit and π_3 has the smallest. Results in Fig. 8.3b and 8.3c match the analysis quite well, e.g., Fig. 8.3b shows that BHR increases by 21% for π_1 and 11% for π_3, respectively. Eq. (8.4) suggests given the same cache model and caching algorithm, the improvement ratio for these distributions should equal to their partial caching benefit ratio. The result shows these two values are indeed very close to each other ($\Delta_{\pi_1,1} : \Delta_{\pi_2,1} : \Delta_{\pi_3,1} \approx 2 : 1.4 : 1$). Compared with integral caching, chunking also helps FPR improve from -4% to 6%. The reason for a negative FPR is that the model searches for content in the whole network which may lead to fetching content from somewhere further than CP. The results show that the closer the chunk popularity is to a uniform distribution (π_3), the less benefits partial caching brings.



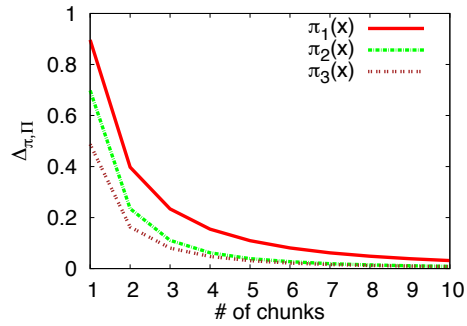
(a) Internal popularity as a function of offset.



(b) BHR as a function of # of chunks.



(c) FPR as a function of # of chunks.



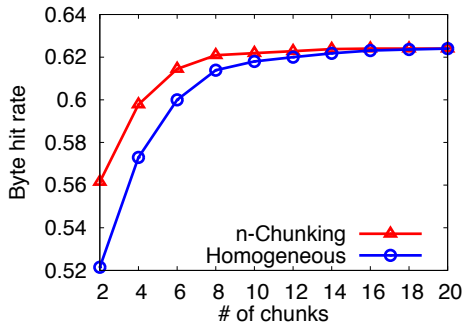
(d) Popularity distance as a function of # of chunks.

Figure 8.3: Effect of chunk popularity on BHR and FPR in dynamic model.

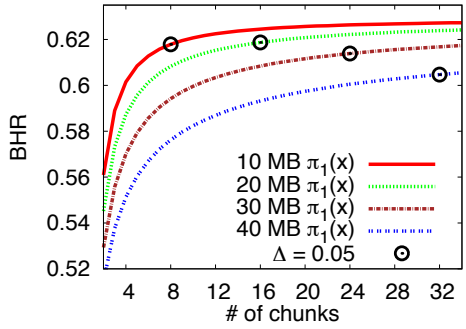
Although $\Delta_{\pi,1}$ can accurately predict the potential benefit, it does not measure how fast it vanishes. Fig. 8.3d plots the popularity distance Δ_{π_i,Π_i} as a function of number of chunks and shows that a few chunks can significantly reduce popularity distance Δ_{π_i,Π_i}. This is also visible in Figs. 8.3b and 8.3c which show that the metric in question improves fastest when the number of chunks is small. Adding chunks beyond 10 yields negligible additional benefits. The slower the convergence (π_1), the larger the benefits from partial caching.

n-Chunking vs. Homogeneous Chunking

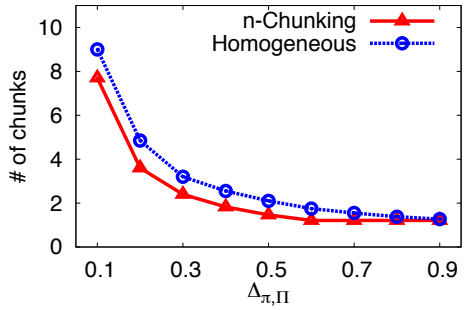
As mentioned in Section 8.3, the differences between chunking schemes diminish as the number of chunks increases. Fig. 8.4a shows the performance of our n-Chunking scheme and a homogeneous chunking scheme, and illustrate that the difference disappears when there are 20 chunks. Fig. 8.4b shows how BHR changes as a function of number of chunks for different



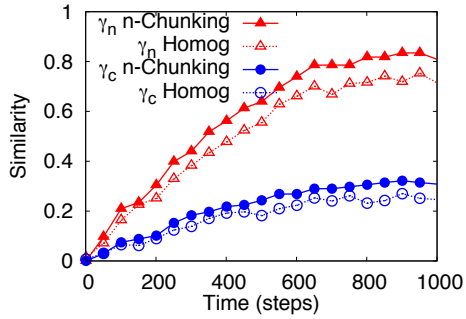
(a) BHR as a function of # of chunks.



(b) BHR under different average file size.



(c) # of chunks as a function of Δ.



(d) Similarity as a function of # of timesteps.

Figure 8.4: Comparison of n-Chunking and homogeneous chunking. Similarity metrics between dynamic and static caching.

file sizes. The three black circles on each line mark the number of chunks needed by n-Chunking algorithm to achieve $\Delta_{\pi,\Pi} = 0.1$. Note that this number increases linearly as file size grows, i.e., $\Delta_{\pi,\Pi} = 0.1$ is achieved at exactly same chunk size (0.8 MB). It verifies our analysis in Section 8.3, showing that average chunk size is a key factor of performance gain. The results also indicate that the speed at which partial caching benefit decays will decrease as file grows because we need more chunks to achieve the same gain. Increasing file size also increases the number of chunks and even though they follow the same popularity distribution, the tail becomes heavier and degrades caching performance.

We also compared the number of chunks a scheme will generate to achieve the target $\Delta_{\pi,\Pi}$. In the experiment, we use Weibull distribution with different parameters to model the internal popularity of a 10 MB file ($\lambda \in [1, 5]$, $k \in [0.5, 5]$). Fig. 8.4c shows that for a small $\Delta_{\pi,\Pi}$, n-Chunking always uses less chunks, with about 40%–50% improvement compared with homogeneous chunking. However, as $\Delta_{\pi,\Pi}$ increases, two schemes eventu-

ally become the same since one or two chunks are sufficient to achieve the goal.

In Section 8.4 we defined both a dynamic (DyPa) and a static (StPa) caching strategy. Intuitively, DyPa should be more effective and our results confirm this. For space reasons, we present only the cache- and network-level similarities between these two approaches. Fig. 8.4d measures what ratio of the content in DyPa is the same as that in StPa in each step. Initially the similarity is low, but as the simulation runs longer, network-level similarity increases, indicating that they store similar content. A partial explanation is that we do not consider changes to content popularity, thus this result is to be expected. However, the cache-level similarity remains very low indicating that content placement is very different in the two cases. Fig. 8.4d also shows that n-Chunking increases the similarity at both network and cache level.

8.5 Low-Complexity Partial Caching

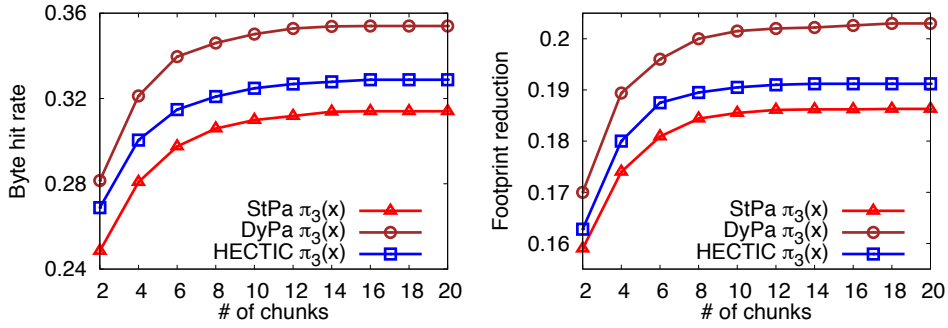
We now propose a low-complexity heuristic caching strategy combined with naive homogeneous chunking at the server and evaluate its performance by comparing against the optimal solution on tree topologies. We then compare against other solutions in the literature on the realistic ISP topologies. The results are from the emulation experiments on our testbed.

HECTIC: HighEst CosT Item Caching

HECTIC is a low-complexity caching strategy devised to achieve the objectives in Section 8.4. To illustrate the design rationale of HECTIC, we consider three components – *admission policy*, *eviction policy* and *cooperation policy*.

Admission policy determines which objects are to be cached and which not. But in a network of caches, caches far from clients only receive a filtered version of the actual user request pattern because of the hits in the caches closer to the client. This so-called *filtering effect* [77,78] may significantly impact the effectiveness of a hierarchical caching network. In [72] we proposed Cachedbit which spreads content probabilistically in the caches along a path. The caching probability at R_k is the reciprocal of its distance from the client: $p_k = \frac{1}{d_{c,R_k}}$. As we showed in [72], Cachedbit is easy to implement, has low overheads, and provides some immunity against filtering effect.

For eviction policy, we propose a utility-based replacement algorithm.



(a) BHR as a function of # of chunks. (b) FPR as a function of # of chunks.

Figure 8.5: Performance of HECTIC, StPa and DyPa on tree topology.

Let *chunk utility* of chunk $f_{i,j}$ at router R_k be:

$$u_{i,j}^{k} = s_{i,j} p_i p_{i,j} c_{k,M+1} \quad \forall f_{i,j} \in C_k \tag{8.15}$$

where $c_{k,M+1}$ represents the cost of retrieving this chunk from CP. Utility function (8.15) considers chunk size, its popularity, and the efforts of retrieving it. A cache will evict the chunk with the smallest utility first.

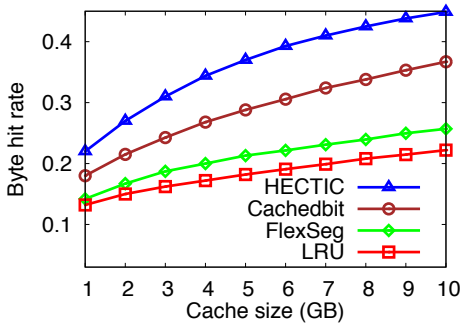
Cooperation between caches can reduce duplicate copies and allow for more efficient use of storage, but at the cost of communication overheads. To reduce these overheads, we reserve one bit in the header to indicate if an object is cached in an upstream cache [72]. This ensures that on a path from the CP to one client, at most one copy of a chunk exists, but paths to other clients may contain additional copies; this improves both BHR and FPR.

HECTIC vs. Optimal Strategies on a Tree Topology

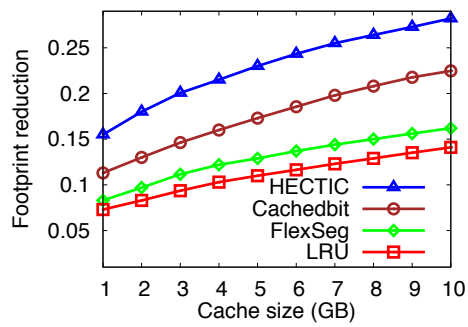
We first compared HECTIC against the dynamic and static optimal caching strategies from Section 8.4 on a 4-level 2-tree topology with parameters as in Section 8.4. Because HECTIC is an en-route caching scheme, we restricted the optimal strategies in being able to retrieve data only from one hop away. Fig. 8.5 depicts BHR and FPR, and shows that HECTIC outperforms static StPa and reaches about 90% of the performance of the dynamic DyPa.

Evaluation with Realistic Settings

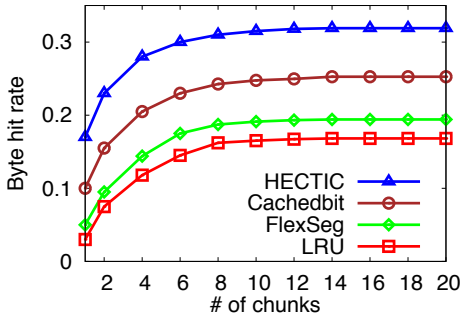
We verified HECTIC's performance in more realistic settings by comparing it on a testbed against several other caching strategies from literature. We



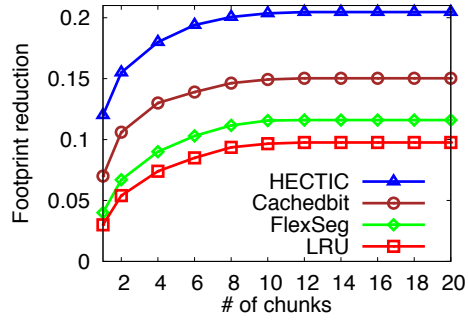
(a) BHR as a function of cache size. 8 chunks per file.



(b) FPR as a function of cache size. 8 chunks per file.



(c) BHR as a function of # of chunks. 3GB storage per node.



(d) FPR as a function of # of chunks. 3GB storage per node.

Figure 8.6: Performance evaluation of different in-network and partial caching strategies on Sprint network.

used four ISP router-level topologies from Rocketfuel [84] project: Exodus, Sprint, AT&T and NTT. We only present results on Sprint network due to space limitations; other networks yielded similar results. All the experiments are performed on our department cluster consisting of 240 Dell PowerEdge M610 nodes equipped with 2 quad-core CPUs, 32GB memory, and connected to a 10-Gbit network. All nodes run Ubuntu SMP with 2.6.32 kernel.

We connected the server to the router with the highest degree and randomly connected clients at 30% of the routers. Each router is equipped with cache size of 3GB. Link cost and latency between two routers were set according to the topology traces, which reflects the realistic values.

We used the gravity model for generating the traffic patterns. In the gravity model, we first map a client to the city according to its access router and traffic from the client is proportional to the city's population. The client in the city with the smallest population sends out 1 million requests and traffic from other clients is scaled up proportionally based on the city population.

For content, we used the real trace from Cha et al. [83]. We selected Youtube Entertainment Category trace which contains 1,687,506 objects. The trace contains video id, length, views, rating and etc. For other content parameters, we set based on investigations on YouTube by [141, 142]. The aggregated video size is 12.87 TB. Work in [83] showed that videos' encoding rates are similar, thereby it is reasonable to assume the file size is proportional to the video length. In [141], Cheng et al. showed that the average file size is 8.4 MB. Based on these results, we set the video size proportional to their length.

Since Heatmap data [34] is not public, we use a Weibull distribution to model file internal popularity. We assume user behavior on different videos differs. To model a heterogeneous access pattern, λ and k are drawn uniformly from $[1, 5]$ and $[0.5, 5]$ respectively, which models the various situations of how popularity varies within a file. All the results are calculated as the arithmetic average of 50 experiments. In each experiment, the client access routers are randomly re-selected to guarantee the results are robust and representative.

We chose another three caching algorithms to compare against HECTIC. LRU is a simple caching strategy with no special admission policy and LRU as replacement policy. Cachedbit is the algorithm presented in [72] and it uses the simple admission policy described in Section 8.5 and LRU as replacement. FlexSeg is a partial caching algorithm presented in [143] where it was also shown to have better performance than other partial

caching algorithms.

Fig. 8.6a and 8.6b show how the performance of three caching strategies change as the cache size increases from 1 GB to 10 GB. Fig. 8.6c and 8.6d show performance change with the increasing number of chunks and keeping the per-cache storage at 3 GB. FlexSeg performs rather poorly, barely beating the simple LRU-based solution, whereas HECTIC shows very good performance. Compared with LRU, FlexSeg has a fine-grained policy to capture the user request pattern. Hence, it performs slightly better, especially in the case of a single cache. However, FlexSeg is designed for edge caches instead of a cache network. Like LRU, it suffers from very low utilization of the aggregate cache storage along the path, which accounts for its poor performance.

Conclusion: Naive chunking at the server and simple caching in the network outperform complex partial caching algorithms running in the network.

8.6 Conclusion

Most of the partial caching work concerns video streaming services [143–148]. In [144], Sen et al. proposed prefix caching for multimedia streams, which caches the initial frames to reduce the start-up latency at client end, and smooth delivery rate variability. Jin et al. [145] take more factors into account like video bit rate, available bandwidth and etc., instead of solely relying on object popularity. However, they did not optimize byte hit rate and footprint reduction, neither take user access pattern into account.

In [146] Yu et al. studied the internal popularity of videos on a streaming website, showing the decline in popularity as the file offset increases. Therefore, they associate videos with a popularity function, indicating the probability of a segment being viewed as inversely proportional to its distance to the beginning. Some algorithms [144,147] are also developed based on the assumption that the earlier segments are more popular than the later ones. However, this assumption does not always hold as [34] clearly shows that internal popularity can be arbitrary mix of non-continuous portions.

All of [32, 143, 147, 149] use (byte) hit rate as one metric to evaluate caching strategies. Proportional Partial Caching (PPC) [32] is designed for P2P video and it tries to infer the content popularity from the request sequence. Under PPC, the cache space allocated for a video grows over time as the number of accesses to that file increases. However, PPC maintains a single variable segment for one file, which reduces the algorithmic complexity but fails to capture the heterogeneous internal popularity. In a similar

vein, algorithm proposed in [149] also allocates cache space proportional to object's popularity, but the popularity is obtained by tracking the bytes played back by all the clients in a time slot.

Similar to our work, [147] uses a priori segmentation and presents two chunking algorithms *pyramid* and *skyscraper*, and an approach of using two caches with different caching policies. Lazy segmentation [150] initially caches the whole object, then determines the segment length at run time. Each object is associated with a function of number of accesses and average time of each access, which also determines the segment length. Similarly to [150], [143] proposes flexible segmentation policy (FlexSeg), which also allows segment size to be adapted at run-time. The difference is that the function which determines the segment size is a function of both frequency and recency of accesses, and FlexSeg is more conservative in allocating space for new objects.

In this chapter, we studied the partial caching by first identifying the origin of the partial caching benefit. Next, we proposed a way to quantify the benefit of chunking and illustrated its relation to the actual performance changes. Based on this analysis, we presented the optimal n-Chunking algorithm and compared it with homogeneous chunking. To evaluate the partial caching, we also developed an optimization model which can be used to calculate the upper bound of in-network caching performance. We devised a low-complexity heuristic, HECTIC, which performs close to the optimal solutions on simple topologies and outperforms existing caching solutions on realistic topologies. Our analytical and experimental results showed that complex partial caching algorithms did not perform as well as a naive homogeneous chunking at the server combined with a simple utility-based caching strategy.

Chapter 9

Fair Collaborative Games

Information-centric networking extensively uses universal in-network caching. However, developing an efficient and fair collaborative caching algorithm for selfish caches is still an open question. In addition, the communication overhead induced by collaboration is especially poorly understood in a general network setting such as realistic ISP and Autonomous System networks. In this chapter, we address these two problems by modeling the in-network caching problem as a Nash bargaining game. We show that the game is a convex optimization problem and further derive the corresponding distributed algorithm. We analytically investigate the collaboration overhead on general graph topologies, and theoretically show that collaboration has to be constrained within a small neighborhood due to its cost growing exponentially. Our proposed algorithm achieves at least 16% performance gain over its competitors on different network topologies in the evaluation, and guarantees provable convergence, Pareto efficiency and proportional fairness.

9.1 Introduction

Due to the shift to content-oriented Internet, Information-Centric Networking (ICN) [1–3] was proposed to ameliorate the pressure on current network infrastructure. ICN architecture extensively uses in-network caching to reduce network traffic and improve content delivery efficiency. Compared to the conventional edge caching, which is usually designed to maximize the local (byte) hitrate, in-network caching is fundamentally different because network topology and cache collaboration play an important role in both algorithm design and system evaluation [46, 151–153]. In other words, simply optimizing local performance in a cache network does not necessarily

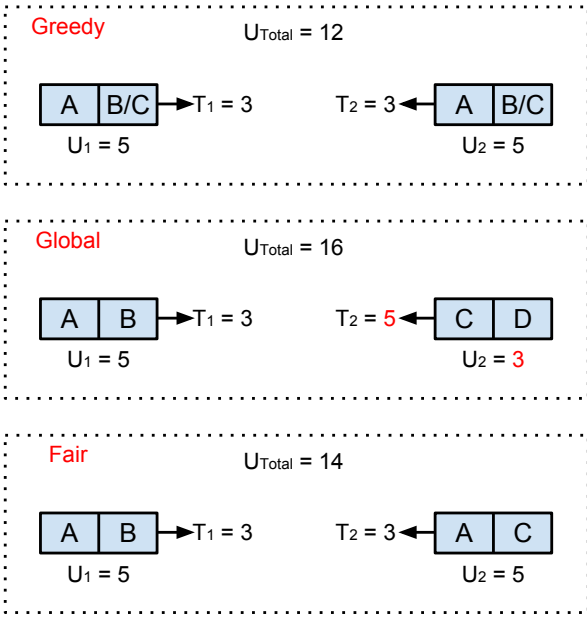
drive the whole system to its optimal state.

As an active research field, there is abundant analytical work in ICN [64, 154–158] which significantly improved the understanding on the functional relation among system performance, traffic flow and network topology. Moving to collaborative caching, similarly, many practical caching algorithms were proposed and analyzed [46,73,153,159–161] in various settings. Nonetheless, two important aspects of future cache networks have long been overlooked.

Firstly, considering that caching is universal and content is pervasive in an ICN context, it is reasonable to assume that multiple Autonomous Systems (AS) with different interest participate in a cache network. Meanwhile, some big content providers like Google and Facebook, along with Akamai, also actively build up a wide range of content distribution networks by connecting to or even directly deploying storage in ISP networks. It is foreseeable that in the near future our core network will transform into a complex content network consisting of heterogeneous caches [162, 163]. The motivation for collaboration is to get additional benefits from others, but caches might be unwilling to sacrifice their own performance for purely altruistic reasons. Even within a single ISP network, where we can reasonably assume all the nodes are obedient, sacrificing certain caches in order to pursue the "global welfare" is not always acceptable or even not safe. Because it may cause severe regional performance problem, especially if the cache is installed at a critical position in the system. The immediate cascading effects can spread fast and wide in the network, further causes much larger damage than anticipated [164, 165]. However, most previous work simply maximizes the aggregated utility under the strong assumption of all others being fully obedient. Consequently, global optimum usually results in performance degradation on certain nodes and might not be acceptable to all nodes. In this chapter, we argue that collaboration should be based on fairness. While global optimum is attractive, it is more important to guarantee that every node will be better off collaborating together than working alone so each part of the network will be improved at the same time and function properly. Rather than simply optimizing the aggregated benefits, we find it more preferable to maximize the additional benefit from collaboration. In other words, we study how to improve the overall system performance without downgrading any individual, as the following example shows.

Example. *We use the mini caching system described in Fig.9.1 as a simple example to illustrate our problem space.*

Case 1: Greedy strategy lets each cache optimize its own performance



Demand Matrix

Object	Demands
A	3
B	2
C	2
D	1

Figure 9.1: A mini caching system consists of two caches, and each can store only two objects. The demand matrix is the same for both caches. The utility is calculated as the amount of demands satisfied. U_{Total} represents the aggregated demands satisfied by the whole caching system. Similarly, U_1 and U_2 represent the demands satisfied by cache 1 and cache 2 respectively. T represents the outgoing traffic due to uncached content. Three caching strategies (Greedy, Global and Fair) are presented.

locally. Because B and C have the same demands, each will be cached with $\frac{1}{2}$ chance, giving an average utility $U_1 = U_2 = 5$ for each cache. The outgoing traffic from each cache is $T_1 = T_2 = 3$. For the whole caching system, the possible content in two caches are $\{A, B; A, B\}$, $\{A, B; A, C\}$, $\{A, C; A, B\}$ and $\{A, C; A, C\}$, each has a probability $\frac{1}{4}$. Therefore, we have the average utility $U_{Total} = 10 \times \frac{1}{4} + 14 \times \frac{1}{4} + 14 \times \frac{1}{4} + 10 \times \frac{1}{4} = 12$.

Case 2: Global strategy tries to maximize the aggregated utility of the whole system. By caching all the objects, the overall cache utilization is improved due to no duplicates in the system, leading to the highest $U_{Total} = 2 \times (3 + 2 + 2 + 1) = 16$. However, the performance of cache 2 drops from $U_2 = 5$ to $U_2 = 3$ comparing to the greedy strategy. Meanwhile, the outgoing traffic from cache 2 also increases by 2, which might cause potential congestion problem in the network.

Case 3: Fair strategy emphasizes the basis of collaboration. Comparing

to the previous two strategies, the overall utility $U_{Total} = 14$ is between the greedy one and global one. Though U_{Total} of a fair strategy is not as high as the global one, we can see the overall system performance is improved and nobody gets worse off because of collaboration.

Secondly, collaboration is mostly achieved by explicitly or implicitly exchanging messages, which inevitably introduces communication overhead. The knowledge of how the collaboration overhead grow on a network is extremely valuable for network researchers and engineers in evaluating communication systems and designing protocols [17]. Nonetheless, the collaboration overhead is either overlooked or overly-simplified in the previous work, and is especially poorly understood on general topologies. In order to simplify the analysis, most previous work introduce a strong assumption on the topological regularity and often use structured networks (e.g. line, tree, grid and etc.) in the cost analysis. Despite of being closed-form, these analytical results in general can hardly be applied to more realistic network settings. Because the topologies in real-life are far from being structured and regular, e.g. ISP networks, ASes topologies and Internet [84]. These realistic networks are usually characterized by their degree distribution and other graph properties such as diameter, centrality, clustering coefficient and etc. Both the wide deployment and the active research of content networks urge us to deepen our understanding on the cost of collaboration and its relation with aforementioned network topological properties.

With an increased interest on collaborative caching and with continuous efforts in deploying various content networks, in this chapter, we investigate above two questions: (1) how to design a collaborative caching strategy which embraces both efficiency and fairness; (2) how much the collaboration overhead costs on general topologies. Specifically, our contributions are as follows

- We formulate the in-network caching problem as a Nash bargaining game. Our solution guarantees provable Pareto efficiency and proportional fairness.

- We derive the functional relation of collaboration overhead on general topologies, and theoretically show the collaboration is practically constrained within a small neighborhood due to its exponential growth in cost.

- We experimentally show the collaboration is highly localized on realistic ISP topologies. The optimal neighborhood is usually less than three hops, and can be further reduced if larger cache is used.

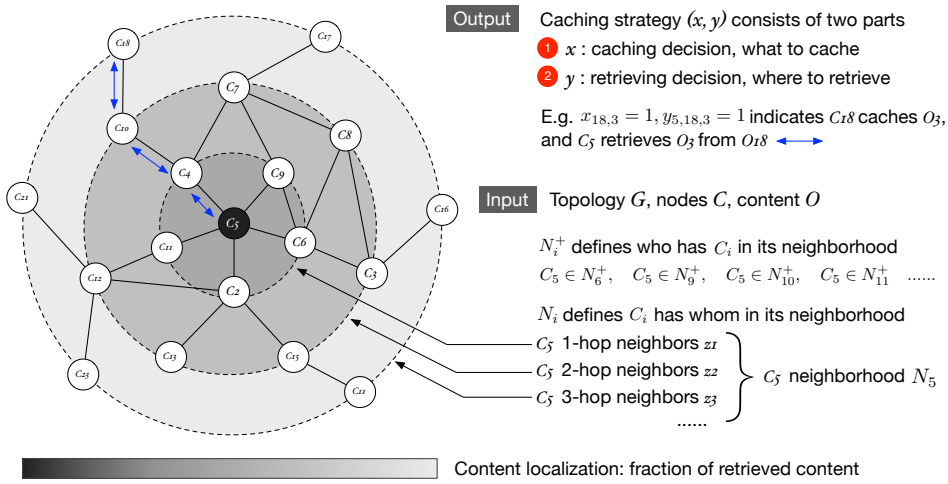


Figure 9.2: A figure illustration of the system model. The grayscale indicates the amount of content retrieved from the neighbors of various distance, which is also an indicator of the intensity of collaboration. Further discussion is in Section 9.8.

- Our results show that while collaborative caching can be beneficial, the benefits only apply when collaborating with a small neighborhood.

9.2 System Model

We assume a content network whose topology can be represented as a graph $G = (V, p)$, where V is the set of nodes characterized with their degree distribution p. p_k denotes the probability that a node has exactly degree k. For each node $v_i \in V$, it is equipped with cache of size C_i. We denote O as the set of content objects. For each $o_k \in O$, we associate two parameters: s_k and $w_{i,k}$. s_k is the object size and $w_{i,k}$ is its aggregated demand (e.g., requests per second) observed from all the clients connected to v_i.

We do not assume that any node has global knowledge of the whole network. Instead, a node is only aware of the information within its neighborhood by collaborating with its neighbors (not necessarily directly connected). Collaboration is characterized by the scope that a node can collaborate with others, namely by its search strength, and we use r_i to represent v_i's search radius measured in hops. r_i uniquely defines a neighborhood for each v_i, which we denote as $N_i = \{v_j | l_{i,j}^* \leq r_i, \forall v_j \in V, v_i \neq v_j\}$, where $l_{i,j}^*$ measures the length of the shortest path between v_i and v_j. Let's

further define $N_i^+ = \{v_j | v_i \in N_j, \forall v_j \in V\}$, which represents the set of nodes who have v_i in their neighborhoods. Apparently, with homogeneous search radius, we have $\forall v_i, v_j \in V, r_i = r_j \iff \forall v_i \in V, N_i = N_i^+$. Allowing heterogeneous search radius indicates the neighborhood relation is not symmetric, so N_i and N_i^+ may not be the same in most cases.

Assume that there is a distributed/centralized caching algorithm to manage these networked caches. Such an algorithm is also referred as a caching strategy which can be decomposed into "caching decision" and "retrieving decision". These two parts solve "what to cache" and "where to fetch" respectively. To model such caching strategy, we use two vector decision variables: $\mathbf{x}$ and $\mathbf{y}$. $x_{i,k} \in \{0, 1\}$ denotes whether v_i caches o_k, and $y_{i,j,k} \in \{0, 1\}$ denotes whether v_i retrieves the object o_k from v_j. In the model, we relax the integer constraints on $\mathbf{x}$ and $\mathbf{y}$ to allow both to be real values. Due to the nature of one-dimension *Bin Packing Problem*, the relaxation renders only one fractional object per cache [166]. Considering the total number of cached objects is big, the induced impact on a cache from one partial object is almost negligible, especially when the object is not the most popular one. Therefore, such relaxation provides a tight and optimistic bound of the original *0-1 integer programming* problem and also significantly simplifies our analysis. Besides, it leads to a even better intuitive explanation since a content file is usually divided into many smaller pieces (i.e. chunks) in practice to improve the transmission efficiency. Allowing real values makes it possible to represent that only a fraction of the file is cached or retrieved therefore the model is more realistic. For a partial object, the beginning of a fraction is always at the zero offset in a file, further discussion on this can be found in Section 9.8. Because a caching strategy is essentially a mapping which by definition can be viewed as a function of its subscript, we have the following definition.

Definition 1. *A caching strategy for a network G is a tuple of functions $(\mathbf{x}, \mathbf{y})$ where $\mathbf{x} : V \times O \to [0, 1]$ and $\mathbf{y} : V \times V \times O \to [0, 1]$. The family of all such tuples is denoted as Ψ, which represents the whole space of all caching strategies.*

Definition 2. *A caching strategy for a node v_i is defined as $(\mathbf{x_i}, \mathbf{y_i})$, where $\mathbf{x_i} : \{v_i\} \times O \to [0, 1]$ and $\mathbf{y_i} : \{v_i\} \times V \times O \to [0, 1]$ are the partial functions of $\mathbf{x}$ and $\mathbf{y}$ with domains restricted to $\{v_i\} \times O$ and $\{v_i\} \times V \times O$ respectively.*

Note "$\times$" above represents Cartesian product when applying to sets. For the content that a node cannot store due to its capacity limit, it may try to fetch them from nearby neighbors. Therefore a node can always get some extra benefit by collaborating, and it would be beneficial if such utility

is maximized. From a practical perspective, an optimal caching strategy is considered a good strategy if we have

1. Pareto efficiency is achieved in the collaboration.
2. Well-defined fairness is achieved among the nodes.

These two requirements are proposed based on the following considerations. First, as system resources are scarce and valuable, being Pareto efficient guarantees no system resource is wasted. However, Pareto efficient solution may not be unique in vector optimization. From an individual node's perspective, one important motivation to collaborate is obtaining extra benefit. As we have argued, it is hard to justify that a node is altruistic and willing to sacrifice his own performance for a global optimum. Second requirement emphasizes that maximizing the utility from collaboration should not hurt individual performance, so a certain well-defined fairness must be achieved.

9.3 Formulation in Bargaining Framework

Bargaining game is a game theoretical model for analyzing how players collaborate to allocate certain shared resource. The process of collaboration is called bargaining. If the agreement cannot be reached during bargaining, the situation is referred as negotiation breakdown. The original bargaining game is a two-player game, but it can be easily extended to multiple players.

In a bargaining game, there can be multiple Pareto efficient solutions. Nash proved [167] that there is only one unique solution which satisfies all the four axioms as follows: (1) Pareto optimality; (2) Scale invariance; (3) Symmetry; (4) Independence of the irrelevant alternatives. Such a solution is called Nash bargaining solution (NBS). NBS is an axiomatic solution and is agnostic about the actual mechanism through which the agreement is reached. Instead, it only concerns the eventual outcome of a bargaining process by solving the following optimization problem.

$$\max \prod_{v_i \in V} (U_i - u_i^0) \tag{9.1}$$

Eq.(9.1) is called Nash product. u_i^0 is the initial disagreement value for the player i. The disagreement value is defined as the worst payoff a player would accept, any value lower than that will break down the negotiation. Please refer to [168] for more details on bargaining games.

A node serves client requests by storing popular content in local cache. Due to its storage capacity limit, the local cache needs to be used wisely. However, a node's utility can be improved with neighbors' help. From network perspective, the aggregated capacity in a cache network is a resource shared by all the nodes. Collaboration thus indicates a node should also take others' needs into account while optimizing its own utility. Practically, this means local caching decision should be made via negotiation. In the following, we give the formal definition of in-network caching game and its solution.

Definition 3. *An in-network caching game is a tuple* (Ω, u^0)*, where* $\Omega \subset \mathbb{R}^{|V|}$ *contains all the utility values obtainable via collaboration,* $u^0 \subset \mathbb{R}^{|V|}$ *contains all the disagreement values leading to a negotiation breakdown.*

In in-network caching context, a node only stops collaborating with others if it cannot be better-off than simply using its own cache. So disagreement value u^0 is easy to estimate. Let $\Omega^e \subset \Omega$ be the Pareto frontier of set Ω, which is a concave function with closed compact convex domain. A game is considered fair iff its outcome is fair. Therefore,

Definition 4. *A fair collaborative game is a game* (Ω, u^0) *with Nash bargaining solution, namely a function* $f : \Omega^e \to \Psi$ *such that* $f(\Omega, u^0) = (\mathbf{x}, \mathbf{y})$ *uniquely maximizes* $\prod_{v_i \in V}(U_i - u_i^0)$.

By definition, the solution satisfies the aforementioned four axioms. Besides, NBS is the only solution that provides proportional gains with respect to the nadir point of the bargaining set [168]; we will discuss this again in Section 9.7.

To solve the problem more efficiently, especially when multiple players get involved, the product of terms is usually translated into its equivalent summation form. By taking logarithm of the objective function (9.1), we have $\ln(\max \prod_{v_i \in V}(U_i - u_i^0)) = \max \ln(\prod_{v_i \in V}(U_i - u_i^0))$. Therefore NBS can also be obtained by solving the following equivalent problem

$$\max \sum_{v_i \in V} \ln(U_i - u_i^0) \tag{9.2}$$

9.4 Structure of Collaboration

We first derive the centralized solution to expose the structure of collaboration, from which we show neighborhood plays a key role in the optimization process. Then we carry on the analysis on communication overhead due to collaboration.

Given node v_i, its utility can be defined as eq.(9.3). But note that the utility is not confined to the following specific form as long as the similar ideas below are described in an affine function.

$$U_i = \sum_{o_k \in O} s_k w_{i,k} x_{i,k} + \sum_{o_k \in O} \sum_{v_j \in N_i} \frac{s_k w_{i,k}}{l^*_{i,j} + 1} y_{i,j,k} \tag{9.3}$$

First term represents the utility gained by locally cached content and the second one represents the utility gained by neighbors' help. The second term also indicates that the gain of retrieving remote content decreases as the distance increases. From a practical perspective, it indicates that a node prefers fetching from the closest source to avoid long delay or extra traffic. Note that any form of eq. (9.3) which is affine is possible, the form above is not the only possibility. Without loss of generality, we assume unit object size $s_k = 1$, also let $l_{i,j} \triangleq l^*_{i,j} + 1$ for simplicity of expression. Plugging in eq.(9.3), then the optimization problem based on the bargaining framework is

$$\max \sum_{v_i \in V} \ln\Big(\sum_{o_k \in O} w_{i,k} x_{i,k} + \sum_{o_k \in O} \sum_{v_j \in N_i} \frac{w_{i,k}}{l_{i,j}} y_{i,j,k} - u^0_i \Big) \tag{9.4}$$

Subject to

$$\sum_{o_k \in O} x_{i,k} \le C_i, \quad \forall v_i \in V \tag{9.5}$$

$$\sum_{v_j \in N_i} y_{i,j,k} \le 1, \quad \forall v_i \in V, \forall o_k \in O \tag{9.6}$$

$$y_{i,j,k} \le x_{j,k}, \quad \forall v_i, v_j \in V, o_k \in O \tag{9.7}$$

$$x_{i,k} \in [0,1], \quad \forall v_i \in V, o_k \in O \tag{9.8}$$

$$y_{i,j,k} \in [0,1], \quad \forall v_i, v_j \in V, o_k \in O \tag{9.9}$$

Constraint (9.5) means the content stored at a node cannot exceed its cache capacity. Constraint (9.7) says v_i can retrieve o_k from v_j only if v_j cached it; it also says v_i cannot get more than v_j can offer. Constraint (9.6) simplifies the data scheduling by constraining a node to retrieve maximum one complete object in a cache period. Constraints (9.8) and (9.9) impose the domain of decision variables. One technical detail needs special caution is the concavity of the object function (9.4). Generally speaking, the composite of a logarithmic function and an arbitrary function does not necessarily preserve concavity. However, Lemma 1 shows the object function under our investigation is indeed concave.

Lemma 1. *The problem (9.4) is a convex optimization problem.*

Proof. The proof is trivial. Since U_i in eq. (9.3) is affine and positive, non-negative weighted sum of U_i is still affine and positive. All the affine functions are log-concave. So the objective function (9.4) is concave.

In addition, all the constraints (9.5)(9.6)(9.7)(9.8) and (9.9) are affine. Therefore, problem (9.4) is a convex optimization problem. □

Theorem 1. *In a fair collaborative game, for the optimal caching strategy* $(\mathbf{x_i^*}, \mathbf{y_i^*})$ *of node* v_i*, there exist non-negative vectors* $\boldsymbol{\alpha} \succeq 0$*,* $\boldsymbol{\beta} \succeq 0$*,* $\boldsymbol{\gamma} \succeq 0$*,* $\boldsymbol{\delta} \succeq 0$ *and* $\boldsymbol{\lambda} \succeq 0$*, such that*

$$x_{i,k}^* = \frac{1}{\alpha_i + \gamma_{i,k} - \sum_{v_j \in N_i^+} \lambda_{j,i,k}} - \frac{\tau_{i,k}}{w_{i,k}} \tag{9.10}$$

$$y_{i,j,k}^* = \frac{1}{\lambda_{i,j,k} + \beta_{i,k} - \delta_{i,k}} - \frac{l_{i,j}\tau'_{i,k}}{w_{i,j}} \tag{9.11}$$

where $\tau_{i,k} = U_i - u_i^0 - w_{i,k}x_{i,k}$ *and* $\tau'_{i,k} = U_i - u_i^0 - \frac{w_{i,k}}{l_{i,j}} y_{i,j,k}$.

Proof. Obviously caching decision space $[0, 1] \subset \mathbb{R}_+$ is a nonempty, compact and convex set. Since the objective function (9.4) is a continuously differentiable concave function, and all the constraints on the variables are affine, Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for the existence of an optimal solution.

To derive the optimum of a function with constraints, we first derive the Lagrangian $\mathcal{L}(\cdot)$ of eq. (9.4). Let $\boldsymbol{\alpha} \succeq 0$, $\boldsymbol{\beta} \succeq 0$, $\boldsymbol{\gamma} \succeq 0$, $\boldsymbol{\delta} \succeq 0$ and $\boldsymbol{\lambda} \succeq 0$ be the KKT multipliers associated with constraints. Their subscripts are self-explained by the corresponding constraints associated with. Then we have

$$\begin{aligned}
&\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\delta}) = \\
&\sum_{v_i \in V} \ln(U_i - u_i^0) - \sum_{v_i \in V} \sum_{v_j \in N_i} \sum_{o_k \in O} \lambda_{i,j,k}(y_{i,j,k} - x_{j,k}) \\
&- \sum_{v_i \in V} \alpha_i \Big(\sum_{o_k \in O} x_{i,k} - C_i \Big) - \sum_{v_i \in V} \sum_{o_k \in O} \beta_{i,k} \Big(\sum_{v_j \in N_i} y_{i,j,k} - 1 \Big) \\
&- \sum_{v_i \in V} \sum_{o_k \in O} \gamma_{i,k}(x_{i,k} - 1) + \sum_{v_i \in V} \sum_{v_j \in N_i} \sum_{o_k \in O} \delta_{i,j,k} y_{i,j,k}
\end{aligned}$$

Note we dropped constraints $x_{i,j} \geq 0$ and $y_{i,j,k} \leq 1$ in making the Lagrangian because constraints (9.6) and (9.7) make them redundant. In the following derivation, we let $\tau_{i,k} = U_i - u_i^0 - w_{i,k}x_{i,k}$ and $\tau'_{i,k} = U_i - u_i^0 -$

$\frac{w_{i,k}}{l_{i,j}} y_{i,j,k}$ for the simplicity of representation. For the objective function to reach its optimum, first order necessary and sufficient conditions are

$$\begin{aligned}
&\nabla \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\delta}) = 0 \\
&\iff \frac{\partial \mathcal{L}}{\partial x_{i,k}} = 0, \forall v_i, v_j \in V, \forall o_k \in O \\
&\iff \frac{w_{i,k}}{U_i - u_i^0} + \sum_{v_j \in N_i^+} \lambda_{j,i,k} - \alpha_i - \gamma_{i,k} = 0 \\
&\iff x_{i,k}^* = \frac{1}{\alpha_i + \gamma_{i,k} - \sum_{v_j \in N_i^+} \lambda_{j,i,k}} - \frac{\tau_{i,k}}{w_{i,k}}
\end{aligned}$$

with complementary slackness

$$\begin{cases}
\lambda_{i,j,k}(y_{i,j,k} - x_{j,k}) = 0, & \forall v_i, v_j \in V, \forall o_k \in O \\
\alpha_i(\sum_{o_k \in O} x_{i,k} - C_i) = 0, & \forall v_i \in V, \forall o_k \in O \\
\beta_{i,k}(\sum_{v_j \in N_i} y_{i,j,k} - 1) = 0, & \forall v_i \in V, \forall o_k \in O \\
\gamma_{i,k}(x_{i,k} - 1) = 0, & \forall v_i \in V, \forall o_k \in O \\
\delta_{i,j,k} y_{i,j,k} = 0, & \forall v_i, v_j \in V, \forall o_k \in O
\end{cases} \tag{9.12}$$

Similarly, we can derive the optimal $y_{i,j,k}^*$ as $x_{i,k}^*$. The optimal caching strategy $(\mathbf{x}^*, \mathbf{y}^*)$ of the network can be derived by solving the equation system (9.12) for all the nodes. □

Theorem 1 exposes the internal structure of collaborations, even though its proof is rather straightforward as above. Calculating $(\mathbf{x_i^*}, \mathbf{y_i^*})$ for node v_i requires the information from $N_i \cup N_i^+$, e.g. $\boldsymbol{\lambda}$, the KKT multiplier associated with constraint (9.7). Actually, the first equation in eq.(9.12) indicates $\boldsymbol{\lambda}$ is the only multiplier shared in neighborhood, others are local variables. $\lambda_{i,j,k}$ can be viewed as the "shadow price" of transferring o_k from v_j to v_i, which is a "cost" for v_i but an "income" for v_j. Thus term $\sum_{v_j \in N_i^+} \lambda_{j,i,k}$ is v_i's total income from serving o_k to those in N_i^+. Eq. (9.10) indicates that if total income due to o_k increases, v_i tends to cache it. Meanwhile, eq. (9.11) suggests that if the cost $\lambda_{i,j,k}$ increases, v_i tends to stop retrieving o_k from v_j. As we can see, the explanation of the results matches our intuition very well.

The whole equation system has $3|O| \times |V|^2 + 2|O| \times |V| + |V|$ variables and same number of equations. The computation overhead can be considerably high if the content set and network are big, which motivates us to look for a more scalable distributed algorithm in Section 9.5. Note though a distributed solution can significantly accelerate the calculations

by parallelism, it will not reduce the overall computation complexity and the performance gain is at the price of increased traffic by exchanging information. The amount of exchanged information will not be less than that in a centralized solution and the collaboration structure are very similar. The growth of such communication overhead of a distributed solution will be thoroughly analyzed on various networks in Section 9.6.

9.5 Distributed Fair In-Network Caching Solution

A centralized solution has several obvious drawbacks in its actual use. First, it suffers from high computation complexity even with moderate problem size. Second, it is not robust enough due to its single point failure. Third, it is not adaptive enough under network dynamics. Hence we need to translate the centralized solution into a distributed one by decomposition techniques. In this section, we show how to derive the distributed solution from the equivalent dual problem and present our Fair In-Network Caching (FIN) algorithm.

To solve an equation system, each node can be viewed as a subsystem. If they simply optimize locally, all the calculations in each subsystem are independent from those in other subsystems. However, variables and constraints due to collaboration make such calculations dependent, therefore they couple a subsystem with others. Such variables and constraints are referred as *complicating variables and constraints* [169].

As discussed in Section 9.4, constraint (9.7) is the only complicating constraint coupling a node with its neighbors. To decompose the objective function, we first rewrite original problem (9.2) into its equivalent convex form.

$$\min - \sum_{v_i \in V} \ln(U_i - u_i^0) \tag{9.13}$$

Then we apply Lagrangian dual relaxation. Lagrangian dual relaxation provides a non-trivial lower-bound of primal; the difference between the dual and the primal is called *duality gap*. In some cases, duality gap can be zero if certain conditions are met as we show below. The Lagrangian

$\mathcal{L}(\cdot) : \mathbb{R}^{2|O||V|^2} \rightarrow \mathbb{R}$ associated with objective (9.13) is defined as follows

$$\begin{aligned} &\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) \\ &= -\sum_{v_i \in V} \ln(U_i - u_i^0) + \sum_{v_i \in V} \sum_{v_j \in N_i} \sum_{o_k \in O} \lambda_{i,j,k}(y_{i,j,k} - x_{j,k}) \\ &= \sum_{v_i \in V} [-\ln(U_i - u_i^0) + \sum_{v_j \in N_i} \sum_{o_k \in O} \lambda_{i,j,k}(y_{i,j,k} - x_{j,k})] \end{aligned} \tag{9.14}$$

$\boldsymbol{\lambda} \succeq 0$ is the dual variable associated with eq.(9.13). Then the Lagrangian dual function $d(\cdot) : \mathbb{R}^{|O||V|^2} \rightarrow \mathbb{R}$ is as follows

$$d(\boldsymbol{\lambda}) = \inf_{\mathbf{x} \in X, \mathbf{y} \in Y} \mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) \tag{9.15}$$

Given $\boldsymbol{\lambda}$, let $\mathbf{x}^*$ and $\mathbf{y}^*$ be the unique minimizers for the Lagrangian (9.14) over all $\mathbf{x}$ and $\mathbf{y}$. Then the dual function (9.15) can be rewritten as $d(\boldsymbol{\lambda}) = \mathcal{L}(\mathbf{x}^*, \mathbf{y}^*, \boldsymbol{\lambda})$. By maximizing the dual function, we can reduce the duality gap. The Lagrangian dual problem of the primal (9.13) is defined as follows

$$\max_{\boldsymbol{\lambda} \in \mathbb{R}^{|O||V|^2}} d(\boldsymbol{\lambda}) = \mathcal{L}(\mathbf{x}^*, \mathbf{y}^*, \boldsymbol{\lambda}) \tag{9.16}$$

The constraints for the dual problem are the same as those of the primal except constraint (9.7) which is already included in the dual objective function. Obviously there must exist a solution $(\mathbf{x}, \mathbf{y}) \in \mathbf{relint}(D)$ which satisfies all the constraints. Also because the objective function (9.13) is convex and all the constraints (9.5)(9.6)(9.8) and (9.9) are affine, Slater's condition holds, and the duality gap is zero. Thus when the dual problem (9.16) reaches its maximum, the primal problem also reaches its minimum. The optimal solution for primal problem (9.13) can be derived from the optimal solution for dual problem (9.16).

As we have shown, a node subsystem can be successfully decoupled from the others in the same neighborhood with Lagrangian dual decomposition. Each node v_i now only needs to optimize its utility locally for a given $\boldsymbol{\lambda}$ by calculating

$$\begin{aligned} &\min \mathcal{L}_i(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) \\ &= -\ln(U_i - u_i^0) + \sum_{v_j \in N_i} \sum_{o_k \in O} \lambda_{i,j,k}(y_{i,j,k} - x_{j,k}) \end{aligned}$$

To help dual problem converge to its optimum, we can use standard projected subgradient method [170] to derive the distributed collaborative

Input:
    Demand matrix $\mathbf{w}$
    Dual variables $\boldsymbol{\lambda}$
Output:
    Caching decision $\mathbf{x}_i$
    Collaboration decision $\mathbf{y}_i$
while $k < k_{stop}$ **do**
  $\mathbf{x}_i, \mathbf{y}_i = \arg\min_{\mathbf{x},\mathbf{y}} \mathcal{L}_i(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda})$
  $\mathbf{h} = \mathbf{y}_i - \mathbf{x}_i$
  for $v_j \in N_i$ **do**
    Retrieve $\mathbf{h}_j$ from v_j
    $\mathbf{h} = \mathbf{h} + \mathbf{h}_j$
  end for
  $\boldsymbol{\lambda} = (\boldsymbol{\lambda} + \xi\mathbf{h})_+$
  $k{+}{+}$
end while

Algorithm 8: Fair in-network caching (FIN) algorithm on v_i

caching algorithm. Let $h(\boldsymbol{\lambda})$ and $\partial d(\boldsymbol{\lambda})$ denote the subgradient and subdifferential of dual function $d(\cdot)$ at point $\boldsymbol{\lambda}$ respectively. Then for every $h_{i,j,k} \in h(\boldsymbol{\lambda})$ we have

$$h_{i,j,k} = y^*_{i,j,k} - x^*_{j,k} \Longrightarrow h(\boldsymbol{\lambda}) \in \partial d(\boldsymbol{\lambda})$$

Vector $\mathbf{h} = h(\boldsymbol{\lambda})$ points to the direction where $d(\cdot)$ increases fastest. In each iteration, node v_i needs to solve the subsystem (9.17) to update dual variable $\boldsymbol{\lambda}$. k represents the k^{th} iteration. ξ_k is the step-size in the k^{th} iteration which can be determined by several standard methods [170]. Projected subgradient method projects $\boldsymbol{\lambda}$ on its constraint ($\boldsymbol{\lambda} \succeq 0$) in each iteration, and we use $(\cdot)_+$ as a shorthand for the Euclidean projection of a point on $\mathbb{R}_+^{|O||V|^2}$. Eventually $\boldsymbol{\lambda}^{(k)} \to \boldsymbol{\lambda}^*$ when $k \to \infty$. The primal solution can be constructed from optimum $\boldsymbol{\lambda}^*$. Note that feasibility is not necessarily needed in every iteration.

$$\begin{cases} \mathbf{x}_i^{(k)}, \mathbf{y}_i^{(k)} = \arg\min_{\mathbf{x},\mathbf{y}} \mathcal{L}_i(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}^{(k)}) \\ \mathbf{h}^{(k)} = -(\mathbf{x}_i^{(k)} - \mathbf{y}_i^{(k)}) \\ \boldsymbol{\lambda}^{(k+1)} = (\boldsymbol{\lambda}^{(k)} + \xi_k \sum_{v_j \in N_i \cup \{v_i\}} \mathbf{h}_j)_+ \end{cases} \tag{9.17}$$

Theorem 2. *Algorithm 8 converges to its optimum as the sequence* $\{\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)} \ldots \boldsymbol{\lambda}^{(k)}\}$ *converges, if a diminishing step size is used such that* $\lim_{i\to\infty} \xi_i = 0$ *and* $\sum_{i=1}^{\infty} \xi_i = \infty$.

Proof. To prove convergence, we first prove the gradient of the dual function is bounded by a constant K, namely the dual function $d(\boldsymbol{\lambda})$ is K-Lipschitz continuous. Second, we show that given the diminishing step size, the Euclidean distance between the optimum $d(\boldsymbol{\lambda}^*)$ and the best value $d(\boldsymbol{\lambda}^\circ)$ achieved in all previous iterations converges to zero in limit.

Since the primal (9.13) is strictly convex and all constraints are linear, dual $d(\boldsymbol{\lambda})$ is strictly concave and differentiable.

$$\frac{\partial d(\boldsymbol{\lambda})}{\partial \lambda_{i,j,k}} = y_{i,j,k} - x_{i,k} \Longrightarrow \left| \frac{\partial d(\boldsymbol{\lambda})}{\partial \lambda_{i,j,k}} \right| \leq 1 \tag{9.18}$$

By Mean value theorem, there exists $\mathbf{c} \in (\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ such that

$$d(\boldsymbol{\lambda}) - d(\boldsymbol{\lambda}') = \nabla d(\mathbf{c})^T (\boldsymbol{\lambda} - \boldsymbol{\lambda}') \tag{9.19}$$

By Cauchy–Schwarz inequality, let $n = |O| \times |V|^2$, we have

$$\begin{aligned} \|d(\boldsymbol{\lambda}) - d(\boldsymbol{\lambda}')\|_2 &= \|\nabla d(\mathbf{c})^T (\boldsymbol{\lambda} - \boldsymbol{\lambda}')\|_2 && (9.20)\\ &\leq \|\nabla d(\mathbf{c})\|_2 \|\boldsymbol{\lambda} - \boldsymbol{\lambda}'\|_2 && (9.21)\\ &\leq \sqrt{n} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}'\|_2 && (9.22) \end{aligned}$$

$\|\cdot\|_2$ above denotes the Euclidean norm. Therefore, $d(\boldsymbol{\lambda})$ is K-Lipschitz continuous and Lipschitz constant $K = \sqrt{n}$. Let $\boldsymbol{\lambda}^*$ denote the maximizer of dual function $d(\boldsymbol{\lambda})$, then

$$\begin{aligned} &\|\boldsymbol{\lambda}^{(k+1)} - \boldsymbol{\lambda}^*\|_2^2 = \|(\boldsymbol{\lambda}^{(k)} + \xi_k \mathbf{h}^{(k)})_+ - \boldsymbol{\lambda}^*\|_2^2 && (9.23)\\ &\leq \|\boldsymbol{\lambda}^{(k)} + \xi_k \mathbf{h}^{(k)} - \boldsymbol{\lambda}^*\|_2^2 && (9.24)\\ &= \|\boldsymbol{\lambda}^{(k)} - \boldsymbol{\lambda}^*\|_2^2 + 2\xi_k \mathbf{h}^{(k)T} (\boldsymbol{\lambda}^{(k)} - \boldsymbol{\lambda}^*) + \xi_k^2 \|\mathbf{h}^{(k)}\|_2^2 && (9.25)\\ &\leq \|\boldsymbol{\lambda}^{(k)} - \boldsymbol{\lambda}^*\|_2^2 + 2\xi_k (d(\boldsymbol{\lambda}^{(k)}) - d(\boldsymbol{\lambda}^*)) + \xi_k^2 \|\mathbf{h}^{(k)}\|_2^2 && (9.26) \end{aligned}$$

Inequality (9.24) comes from the fact that projection of a point onto $\mathbb{R}_+^{|O||V|^2}$ makes it closer to the optimal point in $\mathbb{R}_+^{|O||V|^2}$. Apply inequality (9.26) recursively, we have

$$\begin{aligned} &\|\boldsymbol{\lambda}^{(k+1)} - \boldsymbol{\lambda}^*\|_2^2 \leq \\ &\|\boldsymbol{\lambda}^{(1)} - \boldsymbol{\lambda}^*\|_2^2 + 2 \sum_{i=1}^{k} \xi_i (d(\boldsymbol{\lambda}^{(i)}) - d(\boldsymbol{\lambda}^*)) + \sum_{i=1}^{k} \xi_i^2 \|\mathbf{h}^{(i)}\|_2^2 \end{aligned}$$

Because $\|\boldsymbol{\lambda}^{(k+1)}-\boldsymbol{\lambda}^*\|_2^2 \geq 0$ and $\sum_{i=1}^{k} \xi_i > 0$, and let $d(\boldsymbol{\lambda}^\circ) = \max_{0\leq i<k} d(\boldsymbol{\lambda}^{(i)})$, then

$$
\begin{aligned}
2\sum_{i=1}^{k} \xi_i (d(\boldsymbol{\lambda}^*) - d(\boldsymbol{\lambda}^\circ)) &\leq \|\boldsymbol{\lambda}^{(1)} - \boldsymbol{\lambda}^*\|_2^2 + \sum_{i=1}^{k} \xi_i^2 \|\mathbf{h}^{(i)}\|_2^2 \\
\Longrightarrow d(\boldsymbol{\lambda}^*) - d(\boldsymbol{\lambda}^\circ) &\leq \frac{\|\boldsymbol{\lambda}^{(1)} - \boldsymbol{\lambda}^*\|_2^2 + \sum_{i=1}^{k} \xi_i^2 \|\mathbf{h}^{(i)}\|_2^2}{2\sum_{i=1}^{k} \xi_i} \\
\Longrightarrow d(\boldsymbol{\lambda}^*) - d(\boldsymbol{\lambda}^\circ) &\leq \frac{\|\boldsymbol{\lambda}^{(1)} - \boldsymbol{\lambda}^*\|_2^2 + K^2 \sum_{i=1}^{k} \xi_i^2}{2\sum_{i=1}^{k} \xi_i}
\end{aligned}
$$

$d(\boldsymbol{\lambda}^*) - d(\boldsymbol{\lambda}^\circ) \to 0$ if we choose a diminishing step size which lets $\xi_i \to 0$ and $\sum_1^\infty \xi_i = \infty$, then $\frac{\sum_1^\infty \xi_i^2}{\sum_1^\infty \xi_i} = 0$. (e.g. we can let $\xi_i = \frac{\xi_0}{i}$, then $\sum_1^\infty \xi_i = \infty$ and $\sum_1^\infty \xi_i^2 = \frac{\pi^2}{6}$.) Since the duality gap is zero, eventually the primal problem will converge to its optimum when its dual problem converges. □

With Theorem 2, We can easily show the validity of the proposed algorithm 8 by showing FIN converges to the optimum with a decreasing step size. The proof is fairly standard but gives a theoretical guarantee on the convergence of FIN algorithm.

9.6 Complexity Analysis on General Topologies

Collaboration is meant to improve a node's knowledge on the content distribution within its neighborhood, which further helps the nodes make better caching decisions together. However, as there is no free lunch for optimization, the improvement on caching performance is at the price of extra network traffic. The collaboration inevitably introduces communication overhead. However, in the prior research, such overhead are either overlooked or overly simplified by using highly regular structures such as lines and trees. Though the cost analysis can be significantly simplified, the strong assumption on topological regularity is rather disturbing since it prevents us from applying any conclusion to a more general network setting where the topology can be very flexible. So far, the cost of collaboration is especially poorly understood on general network topologies. In this section, we present how we derive the functional relation between the collaboration overhead and the underlying topological structures only using a general graph model $G = (V, p)$ presented in Section 9.2.

Note that even though FIN is used as an example, the analysis in the following generally applies to any collaborative caching algorithm with few

modifications. By investigating the FIN algorithm presented in Section 9.5, we can see that the communication overhead due to calculating $\boldsymbol{\lambda}$ originate from two parts. The first part is induced by replying the queries from the nodes having v_i in their neighborhood, namely N_i^+. The second part is induced by collecting information from the nodes in v_i's own neighborhood, namely N_i. Given the communication overhead is measured by the number of exchanged messages, and the overhead ϕ_i of node v_i can be calculated as

$$\phi_i = c \times |O| \times (|N_i^+| + |N_i|) \tag{9.27}$$

Scalar c in eq. (9.27) represents a constant factor for communication overhead, and can be understood as message size or other protocol-dependent factors. For system level overhead, we have the following theorem.

Theorem 3. *In a network $G = (V, p)$ where node v_i has a neighborhood N_i uniquely determined by its search radius r_i, the system communication overhead Φ due to collaboration for calculating optimal caching strategy equals*

$$\Phi = 2c \times |O| \times \sum_{v_i \in V} |N_i| \tag{9.28}$$

Proof. System level communication overhead is the aggregation of individual overheads from all the nodes, therefore

$$\Phi = \sum_{v_i \in V} \phi_i = c \times |O| \times \sum_{v_i \in V} (|N_i^+| + |N_i|) \tag{9.29}$$

Given $v_j \in N_i$, neighborhood relation can be written as a tuple (v_i, v_j). Calculating $\sum_{v_i \in V} |N_i|$ is equivalent to counting how many tuples there are in the whole system. Obviously, $v_i \in N_j \Leftrightarrow v_j \in N_i^+, \forall v_i, v_j \in V$, i.e., as long as there is a tuple (v_i, v_j) for N_i, there must be a tuple (v_j, v_i) for some N_j^+, and vice versa. Using double counting technique, we can show $\sum_{v_i \in V} |N_i| = \sum_{v_i \in V} |N_i^+|$. Therefore, eq. (9.29) can be rewritten as

$$\begin{aligned} \Phi &= c \times |O| \times (\sum_{v_i \in V} |N_i^+| + \sum_{v_i \in V} |N_i|) \\ &= 2c \times |O| \times \sum_{v_i \in V} |N_i| \end{aligned}$$

Eliminating N_i^+ will greatly facilitate following proofs. □

Clearly, the system level communication overhead is $\Phi = \Theta(|V| \times |N| \times |O|)$. Theorem 3 shows it is a function of the aggregated neighborhood size, so we do not need to consider N_i^+ in the calculation even when search radius is heterogeneous. To focus on the functional relation between overhead and neighborhood, we fix the network size $|V|$ and content set size $|O|$, and let $\theta \triangleq 2c \times |O|$ for the purpose of simplicity.

Lemma 2. *In a network $G = (V, p)$ where a node's average neighborhood size equals $|\overline{N}|$, system communication overhead equals $\Phi = \theta \times |V| \times |\overline{N}|$.*

Proof. It is trivial by noticing $|V| \times |\overline{N}| = \sum_{v_i \in V} |N_i|$. □

For a node v_i, we can organize its neighborhood N_i into r_i concentric circles according to the neighbor's distance to v_i. We denote z_r as the average number of r-hop neighbors on the r^{th} circle. Obviously, $|\overline{N}| = z_1 + z_2 + ... + z_r$.

Theorem 4. *In a random network $G = (V, p)$ where nodes have average search radius r, the induced system overhead $\Delta_r^{r+1}\Phi$ by increasing the average search radius by 1 equals*

$$\Delta_r^{r+1}\Phi = \theta \times |V| \times \left[\frac{z_2}{z_1}\right]^r \times z_1 \tag{9.30}$$

Proof. Lemma 2 shows system overhead is a function of average neighborhood size. Knowing how neighborhood grows as a function of search radius is the first step for the following proof. Calculating z_1, namely its directly connected neighbors, is trivial and equals a node's average degree. Let $\langle k \rangle$ denote the mean of a given degree variable k. Then we have

$$z_1 = \langle k \rangle = \sum_{k=0}^{\infty} k p_k$$

However, calculating z_r ($r \geq 2$) is not as straightforward as z_1 since degree distribution for a node's neighbor is not the same as general degree distribution for the whole network. Let v_j be one of v_i's next-hop neighbors. Actually, v_j's degree distribution q_k is proportional to both v_i's degree and general degree distribution [171]. Since we should not count the link which leads back to v_i, then we have

$$q_k = \mathbf{Pr}[deg(v_j) = k | deg(v_i) = k + 1] = \frac{(k+1)p_{k+1}}{\sum_m m p_m}$$

Therefore, v_j's average degree, or in other words, the average number of emerging links from v_j equals

$$\sum_{k=0}^{\infty} kq_k = \frac{\sum_{k=0}^{\infty} k(k+1)p_{k+1}}{\sum_m mp_m} = \frac{\sum_{k=0}^{\infty} k(k-1)p_k}{\sum_m mp_m} = \frac{\langle k^2\rangle - \langle k\rangle}{\langle k\rangle}$$

Because we did not assume v_j is on any specific concentric circle except $r \geq 2$, we can use the same logic above to calculate arbitrary r-hop neighbors as follows

$$z_r = z_{r-1} \sum_{k=0}^{\infty} kq_k = \frac{\langle k^2\rangle - \langle k\rangle}{\langle k\rangle} z_{r-1} \tag{9.31}$$

From eq. (9.31), we can further calculate $z_2 = \langle k^2\rangle - \langle k\rangle$. As we already know $z_1 = \langle k\rangle$, by applying replacement recursively, we can rewrite eq. (9.31) as

$$z_r = \left[\frac{z_2}{z_1}\right]^{r-1} \times z_1 \tag{9.32}$$

When system increases the average search radius from r to $r+1$, the system overhead increases from Φ' to Φ''. With lemma 2, we can calculate the difference $\Delta_r^{r+1}\Phi$ by

$$\Delta_r^{r+1}\Phi = \Phi'' - \Phi' = \theta \times |V| \times (|\overline{N''}| - |\overline{N'}|) \tag{9.33}$$

$$= \theta \times |V| \times z_{r+1} \tag{9.34}$$

From eq. (9.32) and (9.34), we get eq. (9.30). We do not intend to give a detailed proof due to the space limitations, please refer to [171] which has more thorough discussions on graph topological properties. □

Given a search radius, Theorem 4 shows that the increase in overhead depends on the ratio between the number of two-hop and one-hop neighbors, and it applies to any general network with arbitrary degree distribution. The overhead only converges if there are less two-hop neighbors than first-hop ones, i.e., $\frac{z_2}{z_1} < 1$, which actually implies the graph is not connected and has multiple components [171].

Corollary 1. *In Erdős-Rényi random network $G = (V, p)$, z is the average node degree. The induced system overhead $\Delta_r^{r+1}\Phi$ by increasing average*

search radius by 1 and the overall system overhead Φ *are calculated as*

$$\Delta_r^{r+1}\Phi = \theta \times |V| \times z^{r+1} \tag{9.35}$$

$$\Phi = \theta \times |V| \times \frac{z(1-z^r)}{1-z} \tag{9.36}$$

Proof. In Erdős-Rényi random network, the degree distribution p_k is given by the following formula

$$p_k = \binom{|V|-1}{k} p^k (1-p)^{|V|-k-1}$$

If $|V| \gg kz$, the binomial distribution above converges to the Poisson distribution in its limit.

$$\lim_{|V|\to\infty} p_k = \frac{z^k e^{-z}}{k!}$$

With Dobiński's Formula, the n^{th} moment of a variable with Poisson distribution can be calculated as eq. (9.37) shows. $\left\{ {n \atop k} \right\}$ denotes *Stirling numbers of the second kind* [171] which represents the number of ways to partition a set of n objects into k non-empty subsets, and is known for calculating $\langle k^n \rangle$.

$$\langle k^n \rangle = e^{-z} \sum_{k=0}^{\infty} \frac{z^k k^n}{k!} = \sum_{k=1}^{n} \left\{ {n \atop k} \right\} z^k \tag{9.37}$$

From eq. (9.37) and eq. (9.32), we have

$$z_2 = \left\{ {2 \atop 2} \right\} z_1^2 + \left\{ {2 \atop 1} \right\} z_1 - z_1 = z_1^2 \Longrightarrow z_r = z^r \tag{9.38}$$

From Theorem 4 and eq. (9.38), we have eq. (9.35) proved. Eq. (9.38) shows that z_1, z_2, z_3 ... form a geometric series, thus the system overhead Φ can be easily derived by calculating the summation of this series. □

Summary: Theorem 4 conveys an important message on collaborative caching, and shows that the collaboration overhead grows exponentially on general connected topologies. Because most natural graphs like Internet and ISP networks have $\frac{z_2}{z_1} > 1$ [84,171], Theorem 4 means collaboration has to be restricted to a very small neighborhood to keep overhead reasonable. It is also worth noting the conclusion does not depend on a specific utility function but applies to any general optimization process on the graph which requires coordination with neighbors [170].

9.7 Fairness in In-Network Caching Games

Pareto efficiency does not indicate fairness. In this section, we study the fairness in caching games. We consider three well-defined and justified fairness metrics [168,172,173]. As the most important one, the proportional fairness is properly modified to fit into our scenario, and the corresponding proof is provided. Let u_i and u_i^w denote the achieved utility and the worst utility of v_i respectively, then we have the following

Definition 5. *Egalitarian Fairness (EF): Egalitarian fairness is achieved iff* $\forall v_i, v_j \in V$, *we have* $u_i - u_i^w = u_j - u_j^w$.

Definition 6. *Max-Min Fairness (MF): Given a performance metric* $g(v_i)$, *max-min fairness is achieved iff* $(\mathbf{x}^*, \mathbf{y}^*) = \arg\max \min_{\mathbf{x},\mathbf{y}} g(v_i), \forall v_i \in V$.

Definition 7. *Proportional Fairness (PF):* $(\mathbf{x}^*, \mathbf{y}^*)$ *is proportionally fair iff* $\forall (\mathbf{x}, \mathbf{y}) \neq (\mathbf{x}^*, \mathbf{y}^*) \Rightarrow \sum_{v_i \in V} \frac{u_i - u_i^*}{u_i^* - u_i^0} < 0$.

EF pursues the absolutely same amount of improvement on each node, and usually leads to a Pareto inefficient solution. PF and MF are widely used in traffic engineering. MF pursues the fairness which maximizes the node with the worst utility, while PF is a generalization of Kalai–Smorodinsky solution which pursues both proportional improvement on all nodes and maximizing the utility from collaboration [172, 173].

Theorem 5. *In a fair collaborative game* (Ω, u^0), *the optimal caching strategy* $(\mathbf{x}^*, \mathbf{y}^*)$ *achieves PF.*

Proof. Because $(\mathbf{x}^*, \mathbf{y}^*)$ is the optimal caching solution, namely $(\mathbf{x}^*, \mathbf{y}^*) = \arg\max_{\mathbf{x},\mathbf{y}} \sum_{v_i \in V} \ln(u_i - u_i^0)$. Let $f(u) = \sum_{v_i \in V} \ln(u_i - u_i^0)$. For $f(u)$ to reach its maximum, the necessary and sufficient first order condition is $\nabla f^* = 0$. $\forall (\mathbf{x}, \mathbf{y}) \neq (\mathbf{x}^*, \mathbf{y}^*) \Rightarrow \exists \boldsymbol{\lambda} \succ 0$ such that $\lambda_i^{-1} = u_i - u_i^0 > 0$. Then $\forall v_i \in V$ we have

$$\begin{aligned} \nabla f^* - \boldsymbol{\lambda} \prec 0 &\Longrightarrow \frac{\partial f^*}{\partial u_i^*} - \lambda_i < 0 \\ &\Longrightarrow \frac{1}{u_i^* - u_i^0} - \lambda_i < 0 \\ &\Longrightarrow \frac{\lambda_i^{-1}}{u_i^* - u_i^0} - \frac{u_i^* - u_i^0}{u_i^* - u_i^0} < 0 \end{aligned}$$

Sum over all the $v_i \in V$, we have

$$\sum_{v_i \in V} \frac{(u_i - u_i^0) - (u_i^* - u_i^0)}{u_i^* - u_i^0} < 0 \Longrightarrow \sum_{v_i \in V} \frac{u_i - u_i^*}{u_i^* - u_i^0} < 0$$

By definition 7, strategy $(\mathbf{x}^*, \mathbf{y}^*)$ is proportionally fair. □

The original form of PF is very similar to that in our definition 7 except u_i^0 is dropped in the formula, therefore can be viewed as a special case of definition 7 with $u_i^0 = 0$. Instead of copying the exact form, we adapted the definition of PF in our scenario. We argue that the original definition of PF used in traffic engineering (e.g. [172]) is improper in in-network caching context. The reason is due to the key difference between in-network caching and traffic engineering. In traffic engineering, the bandwidth of a flow can reduce to zero. Nonetheless in in-network caching, the worst case would be "stopping collaboration with neighbors but using a node's own local cache", so the utility value shall never reach zero. With the original definition of PF, a NBS only achieves PF when the disagreement point is placed exactly at zero, which indicates "fully obedient" therefore fails to reflect a node's bargaining power (e.g. due to its cache capacity and topological position) and its intrinsic selfishness. The adapted version says, that any change in a proportionally fair caching strategy will be detrimental and cause a decrease in the overall benefit from collaboration.

Theorem 6. *In a fair collaborative game* (Ω, u^0) *with optimal strategy* $(\mathbf{x}^*, \mathbf{y}^*)$*,* EF *is sufficient for* MF*, i.e.* $EF \Rightarrow MF$*.*

Proof. We prove the theorem by contradiction. Let's assume solution $(\mathbf{x}^*, \mathbf{y}^*)$ is egalitarian fair, but not max-min fair. u^* is the corresponding utility value.

Let's further assume another solution $(\mathbf{x}', \mathbf{y}') \neq (\mathbf{x}^*, \mathbf{y}^*)$ which achieves max-min fair, and u' is its utility value. In a fair collaborative game, based on the nature of Nash bargaining framework, both $(\mathbf{x}', \mathbf{y}')$ and $(\mathbf{x}^*, \mathbf{y}^*)$ are Pareto optimal.

By definition, max-min fair solution indicates that

$$\min\{u_i' - u_i^w, ...\} > \min\{u_i^* - u_i^w, ...\}, \quad \forall v_i \in V \tag{9.39}$$

By definition, egalitarian fair solution indicates that

$$\min\{u_i^* - u_i^w, ...\} = u_i^* - u_i^w = u_j^* - u_j^w, \forall v_i, v_j \in V \tag{9.40}$$

(9.39), (9.40) $\Longrightarrow$

$$u_i' - u_i^w \geq u_i^* - u_i^w, \quad \forall v_i \in V \tag{9.41}$$

$$u_i' - u_i^w > u_i^* - u_i^w, \quad \exists v_i \in V \tag{9.42}$$

Inequality (9.42) contradicts with the fact that $(\mathbf{x}^*, \mathbf{y}^*)$ is Pareto optimal. So the assumption does not hold. $(\mathbf{x}^*, \mathbf{y}^*)$ must be both egalitarian fair and max-min fair. I.e. $EF \Rightarrow MF$. □

Theorem 5 guarantees the optimal caching strategy to achieve PF of a broader sense. Though EF is seldom used due to Pareto inefficiency, Theorem 6 guarantees that as long as EF is achieved in a fair game (Ω, u^0), MF is also achieved at the same time.

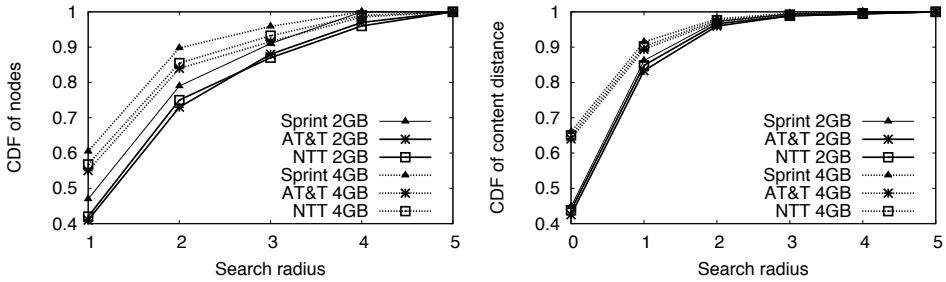
9.8 Numerical Results

We experimented with three ISP topologies (Sprint, AT&T and NTT), and two graph generative models with different parameters: Barabási-Albert (BA) model and Erdős-Rényi (ER) model. The configurations are $\{BA_1 : m = 2\}$, $\{BA_2 : m = 4\}$, $\{ER_1 : p = 1.1 \times \log(n)/n\}$ and $\{ER_2 : p = 1.5 \times \log(n)/n\}$ [171]. For content objects, [83] shows that Youtube videos' popularity follows Weibull distribution with shape parameter $k = 0.513$, and the average file size is 8.4 MB. We use these values in our evaluation to capture the characteristics of realistic settings.

Neighborhood Defined by Search Radius

Fig.9.3a plots the cumulative distribution function (CDF) of optimal neighborhood. Surprisingly, though each node's search radius is preset to the network diameter, the actual neighborhood shrinks significantly after convergence. In all ISP networks, over 80% of nodes have a neighborhood of no more than 3 hops. Fig. 9.3b shows the CDF of the distance of retrieving a content measured in number of hops; note the content served by local cache is also included (i.e. $x = 0$). More impressively, at least 60% of the non-local content is served by the directly-connect neighbors in both 2 GB and 4 GB cases; only minuscule amount is retrieved from those neighbors further than 2 hops. The result also indicates the optimal neighborhood gets even smaller with larger caches.

Fig. 9.4 plots a heatmap of the percent of served content as a function of both search radius and cache size on Sprint network. x-axis is the cache size and y-axis is the search radius, numbers on the grid represent the fraction of the content served. Given a cache size configuration, the fraction



(a) CDF of nodes vs. search radius. (b) CDF of content vs. search radius.

Figure 9.3: Given the initial search radius preset to the network diameter, the neighborhood shrinks to its optimum after convergence. In practice, the optimal neighborhood is small and most content is retrieved from the neighbors within 2 hops.

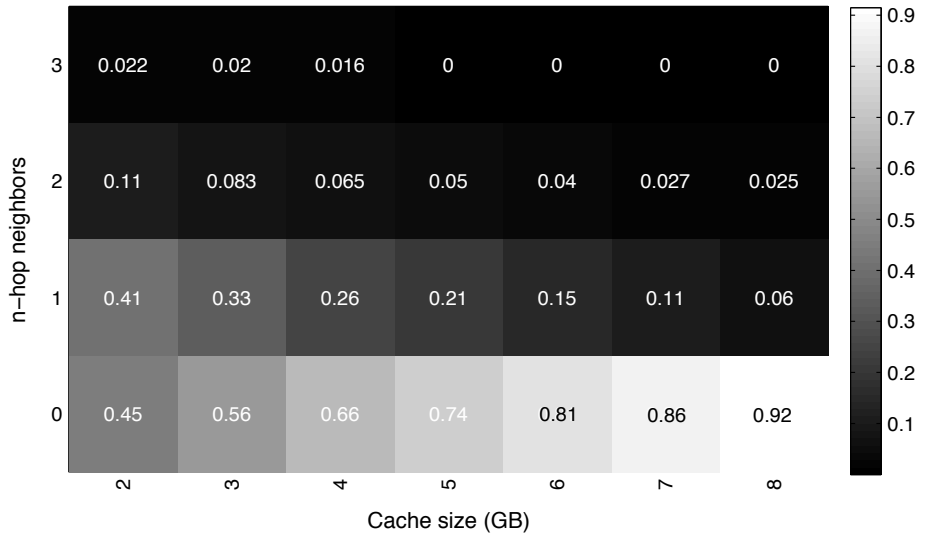


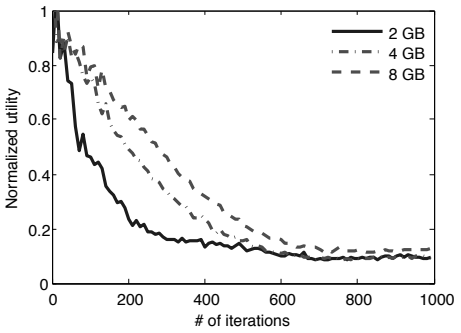
Figure 9.4: Heatmap of content distance and cache size.

ISP network	Sprint	AT&T	NTT
(# of nodes, # of edges)	(604,2279)	(631,2078)	(972,2839)
Overhead growth $\Delta_r^{r+1}\Phi$	$\Theta(5.32^r)$	$\Theta(4.12^r)$	$\Theta(4.51^r)$
Avg. optimal r^* (2GB)	1.83	2.01	2.00
Avg. optimal r^* (4GB)	1.50	1.71	1.65

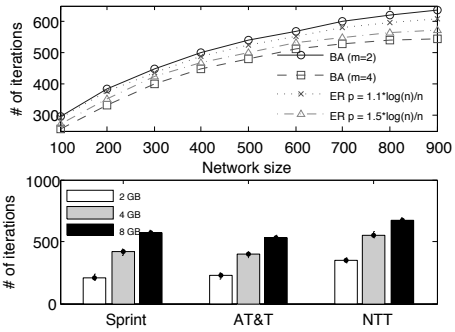
Table 9.1: Growth rate of collaboration overhead $\Delta_r^{r+1}\Phi$ and average optimal search radius r^* on different ISP networks.

of served content drops quickly as distance increases. However, increasing cache size also increases the fraction of locally served content (at $y = 0$), but reduces the need of collaboration. *Further investigation strongly indicates the collaboration is highly localized in a small neighborhood due to the highly skewed content popularity distribution.* In other words, if non-local content is popular enough to be worth fetching remotely, it is highly likely to discover it in the nearby neighbors. Inspired by the observation above, instead of letting the neighborhood shrink to its optimum in optimization process, we let the neighborhood grow step by step in the actual FIN algorithm implementation. The neighborhood growth stops when there is no further benefits. This mechanism can save us from the traffic burst due to exchanging messages in the beginning phase of the algorithm. Table 9.1 summarizes the results on three ISP networks. Though interesting, thorough study on the relation between content and topology is beyond the scope of this thesis and is reserved as future study.

Content overlapping calculates the percent of same content in two different caches, it is an indicator of content diversity in cache networks. We also examined the average content overlapping among the caches and noticed another interesting phenomenon – *content overlapping positively correlates to the cache size configuration.* E.g., the average overlapping is 37.8% for 2 GB cache size configuration, and 62.3% for 4 GB. Namely, there is less content overlapping with small cache size configuration since the nodes need more collaboration from each other to improve their performance. Therefore there is a high degree of content diversity in the neighborhood. With big caches, every node can practically store most of the popular content hence requires less help from the neighbors, which further renders a high degree of content overlapping. In essence, this phenomenon is consistent with our understanding from the experiments in Fig. 9.4.



(a) Aggregated utility as a function of # of iterations on Sprint network.



(b) Convergence rate on both realistic and synthetic networks.

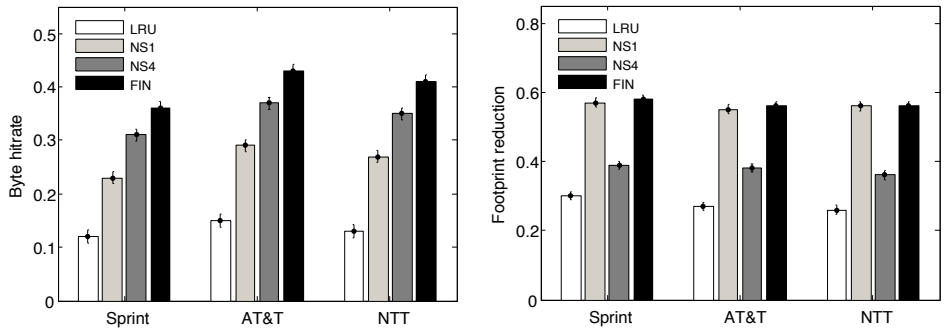
Figure 9.5: Convergence rate of FIN algorithm on both realistic and synthetic networks.

Convergence Rate

Fig. 9.5a shows that the aggregated utility converges as the number of iterations increases on Sprint network. For ease of comparison, the utility has been normalized by its maximum. Larger caches lead to slower convergence rate, because more cached items implies a longer negotiation process among nodes. Given a cache configuration, the convergence rate is influenced by the speed at which information can spread in the network. Upper and lower part in Fig. 9.5b show the convergence rate on both ISP and synthetic networks. As expected, larger ISP networks lead to longer convergence time, but the increase is slower than linear. Similar results were also observed in synthetic ones. Though subgradient method is known for its sensitivity to step size, actually both constant and diminishing step size behaved rather stably in our experiments due to the algorithmic choice on small neighborhoods. Other more robust methods like [169] will be studied in future work.

Caching Performance

To measure caching performance, we use two well-defined metrics byte hitrate (BHR) and footprint reduction (FPR). BHR is a conventional metric to measure saving on inter-domain traffic, while FPR is the reduction on the product of traffic volume and distance which measures saving on network traffic. For comparison, we choose LRU as the baseline, also implement another simple en-route caching heuristic called Nearby Search (NS). NS has a tunable search radius, thus a node can communicate and retrieve



(a) Byte hitrate comparison, 4GB cache.

(b) Footprint reduction comparison, 4GB cache.

Figure 9.6: Performance evaluation of four caching strategies on three realistic ISP networks.

content from other nodes in a neighborhood defined by the radius. NS makes its caching decision independently by optimizing locally instead of via negotiation. We use 1-hop and 4-hop search radius configuration in our experiments, and denote them as NS1 and NS4 respectively.

Fig. 9.6a shows LRU has the worst BHR, whereas our Fair In-Network caching algorithm (FIN; Algorithm 1) has the best. By increasing the search radius, NS4 achieves better BHR, but FIN consistently remains at least 16% better than NS4 over all the networks. Fig. 9.6b shows NS4 has worse FPR (less than 40%) than NS1 and FIN, indicating the gain in BHR is achieved at the price of sacrificing FPR due to increased traffic. NS is still far away from Pareto efficiency despite of being significantly better than LRU. FIN can easily achieve 16% improvement on BHR and 47% on FPR in all the cases. The results indicate FIN reduced much more traffic than other caching strategies and is able to achieve better performance with lower cost.

Further Discussion

Though we explicitly considered the fairness in the modeling part, we implicitly assumed all the participants would run the same prescribed algorithm. In reality, the situation can be different from this assumption. There might be deviant nodes who simply do not run FIN algorithm. In this case, those nodes can be safely excluded from the collaboration without causing any harm to the system since their resources are unavailable. A more troublesome case is that the deviant nodes free-ride their neighbors by being dishonest or refusing to serve. The counterpart can certainly choose to

stop the collaboration if it finds out that no extra benefits can be obtained. Eventually the system will retreat back to the non-collaborative mode if everyone does so. This cascading effect apparently leads to another equilibrium where the whole system suffers from Pareto inefficiency. We do not intend to cover all the possible cases in this short discussion. Designing a sound and complete strategyproof scheme to enforce the obedience is already out of the scope of this thesis and is reserved for the future work.

The linear relaxation in the model is mainly for reducing the complexity in computation and analysis, and it only brings marginal impact on both optimal caching solution and its actual performance. Meanwhile, it leads to an interesting discussion on the partial caching problem on cache networks which heavily relies on the chunk-level modeling. For a partially cached object, assuming that a fraction always starts from the offset zero is equivalent to implicitly assuming that the beginning of a file is more popular than the end. This assumption may hold for certain type of media files like videos [146] but is problematic in general and cannot be applied to arbitrary context without serious justification [174]. Furthermore, if chunk-level popularity is taken into account, how the collaborative caching copes with partial caching is another big question. However, according to our knowledge, the research on chunk-level analysis is severely lacking in the current literature.

9.9 Conclusion

Prior work [154–158] focused on studying the functional relation between system performance and traffic flows to characterize a cache network. Though admission control and replacement policy were explicitly studied on different topologies, collaboration and its related protocol design were mostly overlooked. Recent work indicates two diametrically opposed viewpoints on collaborative caching. On one hand, [4] held a sceptical stance on the general in-network caching approach, [81] further presents the negative result by showing non-collaborative edge caches are sufficient for most of the gains. On the other hand, evidence in [51, 69, 70, 79, 152, 175–177] shows collaboration can indeed improve cache performance. The opposing viewpoints are likely due to the different assumptions in modeling; [81] assumed a strict k-ary tree structure with a single data source at the root, whereas [84] showed assumption of such regular topological structure does not hold in ISP networks. Besides, content is universal and may be retrieved from multiple sources in ICN context [178]. Following this line of research, the recent work [64, 79, 179] focused more on certain system and

design parameters (e.g. topological and routing properties) and investigated their impacts on the effectiveness of collaborative caching in order to gain a holistic understanding.

[46, 64, 73, 79, 151, 153, 159–161, 179, 180] explicitly or implicitly studied the collaborative caching. [73, 151, 160, 161, 180] studied in-network caching in game theory framework by modeling the problem as pure strategic games and analyzed the equilibrium. However, in all these formulations, there is a clear *social optimum* (i.e., the aggregated utility of all nodes) which measures cache system efficiency. The work above also showed this social optimum is seldom reached due to lack of coordination and nodes' inherent selfishness, and the induced inefficiency is quantified with *Price of Anarchy*. Fairness is unfortunately overlooked. Even though fairness has been studied in other context like wireless network and traffic engineering [172, 173], based on our knowledge, there is no prior work studied how fairness should be properly defined on a cache network and how such fairness can be achieved via protocol design. Furthermore, the impact from the network topological properties on algorithm design and caching performance attracts more and more attention in ICN community. Recent work [47, 64, 79, 80, 179] realized the severe limitation of regular topologies and started moving to more general network topological settings. However, the work on the cost analysis of collaborative caching is severely lacking in the new context.

Comparing to the prior work, our work is fundamentally different in three aspects. First, in-network caching problem is modeled as a bargaining game and solved with convex optimization. Second, well-defined fairness is explicitly taken into account in the protocol design. Third, collaboration is carefully defined and the induced overhead is thoroughly analyzed on general topologies.

To summarize, we explicitly defined and studied the fair collaborative games on cache networks. We solved the problem in Nash bargaining framework via convex optimization. Our analysis on collaboration showed its cost grows exponentially whereas the benefit vanishes quickly, therefore collaboration should be constrained to a limited neighborhood. Our proposed FIN algorithm achieved good performance with guaranteed convergence, Pareto efficiency and proportional fairness, on both synthetic and realistic networks. Our results show that while collaborative caching is beneficial, the benefits only apply when collaborating with a small neighborhood.

Chapter 10

Conclusion

10.1 Summary

ICN as a broad research subject covers many topics ranging from content naming, package routing, congestion control, security, energy efficiency and etc. This thesis focused on the in-networking caching, an indispensable core component in the architecture, thoroughly investigated its measurement and design principles, and discussed its connections with content and topology.

We started the thesis with a discussion over miscellaneous metrics which have been or can be potentially introduced into measuring and evaluating an ICN design, with our specific emphasis on the three most important ones pertaining to caching: hit rate, footprint reduction and coupling factor (ordered by the amount of information they contain). Hit rate being the simplest one represents the savings on inter-domain traffic, while footprint reduction represents the savings on intra-domain traffic. Last but not least, coupling factor, as the most complicated metric containing both popularity and topological information, depicts how popular content is distributed in a network, leading another possible categorization of cooperative caching algorithms. Choosing proper and representative metrics, as the initial step for a sound evaluation, is only the beginning of our story. The broad and increasing interest in future network architectures urges our community to develop more thorough measurement methodology with a comprehensive set of metrics.

We then extensively investigated the effects of different cooperative caching heuristics in various settings, showing simple cooperation can already boost caching performance. What's more, we can further enhance content delivery performance by combining proper compacting routing schemes.

With two metrics concerning the network traffic from two different perspectives, namely hit rate and footprint reduction for intra- and inter-domain traffic respectively, we showed there are potentially infinite optimal caching strategies on the Pareto frontier of system caching performance, where cooperation policy determines the final outcomes based on the tradeoff between the two metrics. The process of pushing system performance from under-utilization to Pareto optimum is the process from none/limited cooperation to global cooperation. Bearing the Pareto frontier in mind, we raised the design question that aims at finding a unique caching solution which further incorporates fairness in the design goals, and answered it by modeling in-network caching as a bargaining game. In addition, a careful study on the cooperation overhead was also provided in the thesis.

We consider the work on measurement, evaluation and design of cooperative in-network caching strategies as our major contribution. However, as in-network caching is heavily context-dependent, we are conservative in jumping to any hasty and arbitrary conclusions on the effectiveness of cooperation before a compressive understanding of the future Internet context is achieved and agreed on in the community. As we discussed in the thesis, the actual system performance depends on all three factors: content, topology and cooperation, and no claims solely based on any individual one should be made with confidence.

10.2 Future Direction

Though ICN design brings many advantages in content distribution comparing to the current network architecture, Internet per se is a fast changing ecosystem where current understanding on demands, trends and technologies may be invalidated frequently. Hence whether ICN is going be the solution for the future Internet remains open and needs time to attest. Many research challenges are still awaiting the solutions. Based on the work in this thesis, we list some possible directions in the below:

We proposed in Chapter 5 that greedy routing with hyperbolic embedding can be used to solve mobility and its related issues. We further improved the load balancing on traffic and storage with Prefix-S embedding and topology-aware hashing in Chapter 6. However, being only two possible options for compact routing, other embeddings are also worth investigation.

The metrics used in this thesis are far from complete. As already suggested in Chapter 3, other more complicated ones should also be well examined. E.g. time, as one important dimension to measure system adaptivity, can represent content aging, network evolution, traffic dynamics and

other important information after being integrated into the existing metrics. However, time is seldom taken into account in the existing evaluation work. Another example is energy efficiency, which is attracting more and more interest in ICN research lately. The impacts from incorporating energy metric into the system measurement and design is definitely worth further study.

Last but not least, despite of its resemblance to the pub/sub system, ICN does not use tags to identify a content object. One obvious disadvantage is ICN loses at least one possibility to provide extra information on the content. We believe the future content delivery (including routing, caching and etc.), should better take advantage of the similarities in the content, and even further exploit the semantic structure of the content names. This direction and its related topics surely deserve more research and engineering efforts. ICN may also be qualified as a possible candidate for future semantic web.

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th ACM Conext*. New York, NY, USA: ACM, 2009, pp. 1–12.

[2] Publish/Subscribe Internet Routing Paradigm, "Conceptual architecture of psirp including subcomponent descriptions. Deliverable d2.2, PSIRP project," , August 2008.

[3] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '07. New York, NY, USA: ACM, 2007, pp. 181–192. [Online]. Available: http://doi.acm.org/10.1145/1282380.1282402

[4] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-centric networking: Seeing the forest for the trees," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 1:1–1:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070563

[5] D. Cheriton and M. Gritter, "Triad: A new next-generation internet architecture," July 2000. [Online]. Available: http://gregorio.stanford.edu/triad/

[6] M. Gritter and D. R. 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*, ser. USITS'01. Berkeley, CA,

USA: USENIX Association, 2001, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251440.1251444

[7] B. Baccala, "Data-oriented networking," Internet draft, IETF, Tech. Rep., August 2002.

[8] D. Trossen, G. Parisis, K. Visala, B. Gajic, J. Riihijarvi, P. Flegkas, P. Sarolahti, P. Jokela, X. Vasilakos, C. Tsilopoulos, and S. Arianfar, "PURSUIT Conceptual Architecture: Principles, Patterns and sub-Components Descriptions," Tech. Rep., May 2011.

[9] C. Dannewitz, "Netinf: An information-centric design for the future internet," in *Proc. 3rd GI/ITG KuVS Workshop on The Future Internet*, 2009.

[10] W. K. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G. de Blas, F. Ramon-Salguero, L. Liang, S. Spirou, A. Beben, and E. Hadjioannou, "Curling: Content-ubiquitous resolution and delivery infrastructure for next-generation services," *Communications Magazine, IEEE*, vol. 49, no. 3, pp. 112–120, March 2011.

[11] G. Garcia, A. Beben, F. Ramon, A. Maeso, I. Psaras, G. Pavlou, N. Wang, J. Sliwinski, S. Spirou, S. Soursos, and E. Hadjioannou, "Comet: Content mediator architecture for content-aware networks," in *Future Network Mobile Summit (FutureNetw), 2011*, June 2011, pp. 1–8.

[12] N. B. Melazzi, "Convergence: extending the media concept to include representations of real world objects," in *The Internet of Things*. Springer, 2010, pp. 129–140.

[13] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "Lipsin: Line speed publish/subscribe inter-networking," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 195–206. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592592

[14] A. Carzaniga, M. Rutherford, and A. Wolf, "A routing scheme for content-based networking," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 2, March 2004, pp. 918–928 vol.2.

[15] A. Anand, F. Dogar, D. Han, B. Li, H. Lim, M. Machado, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste, "Xia: An architecture for an evolvable and trustworthy internet," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070564

[16] M. Brunner, H. Abramowicz, N. Niebert, and L. M. Correia, "4ward: a european perspective towards the future internet," *IEICE transactions on communications*, vol. 93, no. 3, pp. 442–445, 2010.

[17] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *Communications Magazine, IEEE*, vol. 50, no. 7, pp. 26–36, July 2012.

[18] M. Bari, S. Chowdhury, R. Ahmed, R. Boutaba, and B. Mathieu, "A survey of naming and routing in information-centric networks," *Communications Magazine, IEEE*, vol. 50, no. 12, pp. 44–53, December 2012.

[19] G. Tyson, N. Sastry, I. Rimac, R. Cuevas, and A. Mauthe, "A survey of mobility in information-centric networks: Challenges and research directions," in *Proceedings of the 1st ACM Workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications*, ser. NoM '12. New York, NY, USA: ACM, 2012, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2248361.2248363

[20] J. Choi, J. Han, E. Cho, T. Kwon, and Y. Choi, "A survey on content-oriented networking for efficient content delivery," *Communications Magazine, IEEE*, vol. 49, no. 3, pp. 121–127, March 2011.

[21] S. Paul, J. Pan, and R. Jain, "Architectures for the future networks and the next generation internet: A survey," *Comput. Commun.*, vol. 34, no. 1, pp. 2–42, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2010.08.001

[22] T. E. M. Service, "Tibco," 1986.

[23] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, "Naming in content-oriented architectures," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ser. ICN '11. New York, NY, USA: ACM, 2011, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2018584.2018586

[24] D. Smetters and V. Jacobson, "Securing network content," Technical report, PARC, Tech. Rep., 2009.

[25] L. Wang, O. Waltari, J. Kangasharju, and U. G. R. Protocol, "Mobiccn: Mobility support with greedy routing in content-centric networks," in *IEEE Global Communications Conference*, 2013.

[26] S. Roos, L. Wang, T. Strufe, and J. Kangasharju, "Enhancing compact routing in ccn with prefix embedding and topology-aware hashing," in *Proceedings of the 9th ACM Workshop on Mobility in the Evolving Internet Architecture*, ser. MobiArch '14. New York, NY, USA: ACM, 2014, pp. 49–54. [Online]. Available: http://doi.acm.org/10.1145/2645892.2645900

[27] IETF Decoupled Application Data Enroute (DECADE) Workgroup, "http://datatracker.ietf.org/wg/decade/," 2011.

[28] Y. Xu, Y. Li, T. Lin, Z. Wang, W. Niu, H. Tang, and S. Ci, "A novel cache size optimization scheme based on manifold learning in content centric networking," *J. Netw. Comput. Appl.*, vol. 37, pp. 273–281, Jan. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2013.03.002

[29] P. Jokela, A. Zahemszky, C. Rothenberg, S. Arianfar and P. Nikander, "LIPSIN: Line Speed Publish/Subscribe Inter-Networking," *ACM SIGCOMM*, 2009.

[30] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, 2011.

[31] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, 1999, pp. 126–134.

[32] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *Networking, IEEE/ACM Transactions on*, vol. 16, no. 6, pp. 1447–1460, 2008.

[33] O. Saleh and M. Hefeeda, "Modeling and caching of peer-to-peer traffic," in *Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on*, Nov 2006, pp. 249–258.

[34] "Wistia," www.wistia.com.

[35] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," *Comput. Netw.*, vol. 57, no. 16, pp. 3128–3141, Nov. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2013.07.007

[36] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: modeling, design and experimental results," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 7, pp. 1305–1314, Sep 2002.

[37] A. Dan and D. Towsley, "An approximate analysis of the lru and fifo buffer replacement schemes," in *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '90. New York, NY, USA: ACM, 1990, pp. 143–152. [Online]. Available: http://doi.acm.org/10.1145/98457.98525

[38] P. Rodriguez, C. Spanner, and E. W. Biersack, "Analysis of web caching architectures: Hierarchical and distributed caching," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 404–418, 2001.

[39] J. Ardelius, B. Grönvall, L. Westberg, and A. Arvidsson, "On the effects of caching in access aggregation networks," in *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*, ser. ICN '12. New York, NY, USA: ACM, 2012, pp. 67–72. [Online]. Available: http://doi.acm.org/10.1145/2342488.2342503

[40] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and storage sharing performance in information centric networking," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ser. IN SIGCOMM ICN'11 Workshop. New York, NY, USA: ACM, 2011, pp. 26–31. [Online]. Available: http://doi.acm.org/10.1145/2018584.2018593

[41] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content-centric networking," in *Proceedings of the 23rd International Teletraffic Congress*, ser. ITC '11. International Teletraffic Congress, 2011, pp. 111–118. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043468.2043487

[42] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and evaluation of ccn-caching trees," in *Proceedings*

of the 10th International IFIP TC 6 Conference on Networking - Volume Part I, ser. NETWORKING'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 78–91. [Online]. Available: http://dl.acm.org/citation.cfm?id=2008780.2008789

[43] Y. Li, H. Xie, Y. Wen, and Z.-L. Zhang, "Coordinating in-network caching in content-centric networks: Model and analysis," in *Proc. IEEE ICDCS*, 2013.

[44] E. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–9.

[45] E. Rosensweig, D. Menasche, and J. Kurose, "On the steady-state of cache networks," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 863–871.

[46] J. Dai, Z. Hu, B. Li, J. Liu, and B. Li, "Collaborative hierarchical caching with dynamic request routing for massive content distribution," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 2444–2452.

[47] L. Wang, S. Bayhan, and J. Kangasharju, "Effects of cooperation policy and network topology on performance of in-network caching," *IEEE Communication Letters*, 2014.

[48] K. Cho, M. Lee, K. Park, T. Kwon, Y. Choi, and S. Pack, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, March 2012, pp. 316–321.

[49] Y. Li, T. Lin, H. Tang, and P. Sun, "A chunk caching location and searching scheme in content centric networking," in *Communications (ICC), 2012 IEEE International Conference on*, June 2012, pp. 2655–2659.

[50] Z. Ming, M. Xu, and D. Wang, "Age-based cooperative caching in information-centric networks," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, March 2012, pp. 268–273.

[51] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proceedings of the*

Second Edition of the ICN Workshop on Information-centric Networking, ser. ICN '12. New York, NY, USA: ACM, 2012, pp. 55–60. [Online]. Available: http://doi.acm.org/10.1145/2342488.2342501

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

[53] X. Tang and S. Chanson, "Coordinated en-route web caching," *Computers, IEEE Transactions on*, vol. 51, no. 6, pp. 595–607, Jun 2002.

[54] Z. Li and G. Simon, "Time-Shifted TV in Content Centric Networks: the Case for Cooperative In-Network Caching," *International Conference on Communications 2011 (ICC'11), Kyoto, Japan*, June 2011.

[55] E. Rosensweig and J. Kurose, "Breadcrumbs: Efficient, best-effort content location in cache networks," in *INFOCOM 2009, IEEE*, April 2009, pp. 2631–2635.

[56] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, and N. Nishinaga, "Catt: Potential based routing with content caching for icn," in *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*, ser. ICN '12. New York, NY, USA: ACM, 2012, pp. 49–54. [Online]. Available: http://doi.acm.org/10.1145/2342488.2342500

[57] Y. Zhu, M. Chen, and N. Akihiro, "Conic: Content-oriented network with indexed caching," *IEEE INFOCOM Workshop*, pp. 1–6, March 2010.

[58] J. Chen, H. Zhang, H. Zhou, and H. Luo, "Optimizing content routers deployment in large-scale information centric core-edge separation internet," *Int. J. Communication Systems*, vol. 27, no. 5, pp. 794–810, 2014.

[59] M. Badov, A. Seetharam, J. Kurose, V. Firoiu, and S. Nanda, "Congestion-aware caching and search in information-centric networks," in *Proceedings of the 1st International Conference on Information-centric Networking*, ser. INC '14. New York, NY, USA: ACM, 2014, pp. 37–46. [Online]. Available: http://doi.acm.org/10.1145/2660129.2660145

[60] A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the scale and performance of cooperative web proxy caching," in *Proceedings of the seventeenth ACM*

symposium on Operating systems principles, ser. SOSP '99. New York, NY, USA: ACM, 1999, pp. 16–31. [Online]. Available: http://doi.acm.org/10.1145/319151.319153

[61] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, "Finding a needle in haystack: Facebook's photo storage," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924947

[62] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 314–329. [Online]. Available: http://doi.acm.org/10.1145/945445.945475

[63] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: Incrementally deployable icn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 147–158. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486023

[64] A. Dabirmoghaddam, M. M. Barijough, and J. Garcia-Luna-Aceves, "Understanding optimal caching and opportunistic caching at "the edge" of information-centric networks," in *Proceedings of the 1st International Conference on Information-centric Networking*, ser. ICN '14. New York, NY, USA: ACM, 2014, pp. 47–56. [Online]. Available: http://doi.acm.org/10.1145/2660129.2660143

[65] C. Imbrenda, L. Muscariello, and D. Rossi, "Analyzing cacheable traffic in isp access networks for micro cdn applications via content-centric networking," in *Proceedings of the 1st International Conference on Information-centric Networking*, ser. ICN '14. New York, NY, USA: ACM, 2014, pp. 57–66. [Online]. Available: http://doi.acm.org/10.1145/2660129.2660146

[66] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, pp. 281–293, June 2000. [Online]. Available: http://dx.doi.org/10.1109/90.851975

[67] L. Zhang, S. Floyd, and V. Jacobson, "Adaptive web caching: towards a new global caching architecture," in *Proceedings of the NLANR Web Cache Workshop*, vol. 97, 1997.

[68] M. Rabinovich, J. Chase, and S. Gadde, "Not all hits are created equal: cooperative proxy caching over a wide-area network," *Computer Networks and ISDN Systems*, vol. 30, no. 22, pp. 2253–2259, 1998.

[69] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson, "Cooperative caching: Using remote client memory to improve file system performance," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI '94. Berkeley, CA, USA: USENIX Association, 1994. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267638.1267657

[70] G. M. Voelker, E. J. Anderson, T. Kimbrel, M. J. Feeley, J. S. Chase, A. R. Karlin, and H. M. Levy, "Implementing cooperative prefetching and caching in a globally-managed memory system," in *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '98/PERFORMANCE '98. New York, NY, USA: ACM, 1998, pp. 33–43.

[71] A. Mislove, A. Post, C. Reis, P. Willmann, P. Druschel, D. S. Wallach, X. Bonnaire, P. Sens, J.-M. Busca, and L. Arantes-Bezerra, "Post: a secure, resilient, cooperative messaging system," in *Proceedings of the 9th conference on Hot Topics in Operating Systems - Volume 9*. Berkeley, CA, USA: USENIX Association, 2003, pp. 11–11.

[72] W. Wong, L. Wang, and J. Kangasharju, "Neighborhood Search and Admission Control in Cooperative Caching Networks," in *Proc. of IEEE GLOBECOM*, December 3-7 2012.

[73] G. Dan, "Cache-to-cache: Could isps cooperate to decrease peer-to-peer content distribution costs?" *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 9, pp. 1469–1482, Sept 2011.

[74] S. Saha, A. Lukyanenko, and A. Yla-Jaaski, "Cooperative caching through routing control in information-centric networks," in *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 100–104.

[75] J. M. Wang, J. Zhang, and B. Bensaou, "Intra-as cooperative caching for content-centric networks," in *Proceedings of the 3rd*

ACM SIGCOMM Workshop on Information-centric Networking, ser. ICN '13. New York, NY, USA: ACM, 2013, pp. 61–66. [Online]. Available: http://doi.acm.org/10.1145/2491224.2491234

[76] K. Katsaros, G. Xylomenos, and G. C. Polyzos, "Multicache: An overlay architecture for information-centric networking," *Comput. Netw.*, vol. 55, pp. 936–947, March 2011. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2010.12.012

[77] C. Williamson, "On filter effects in web caching hierarchies," *ACM Trans. Internet Technol.*, vol. 2, no. 1, pp. 47–77, Feb. 2002. [Online]. Available: http://doi.acm.org/10.1145/503334.503337

[78] R. Fonseca, V. Almeida, M. Crovella, and B. Abrahao, "On the intrinsic locality properties of web reference streams," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1, 2003, pp. 448–458 vol.1.

[79] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache "less for more" in information-centric networks," in *Proceedings of the 11th international IFIP TC 6 conference on Networking - Volume Part I*, ser. IFIP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 27–40.

[80] D. Rossi and G. Rossini, "On sizing ccn content stores by exploiting topological information," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, 2012, pp. 280–285.

[81] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: Incrementally deployable icn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 147–158.

[82] S. K. Fayazbakhsh, A. T. Yin Lin, T. K. Ali Ghodsi, V. S. Bruce M. Maggs, K. C. Ng, and S. Shenker, "Less pain, most of the gain: Incrementally deployable icn," in *SIGCOMM 2013*, August 2013.

[83] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298309

[84] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proceedings of ACM SIGCOMM*, 2002.

[85] E. Eide, "Toward replayable research in networking and systems," in *Archive '10, the NSF Workshop on Archiving Experiments to Raise Scientific Standards*. Salt Lake City, UT: NSF, May 2010.

[86] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*. Boston, MA: USENIX Association, dec 2002, pp. 255–270.

[87] J. Moy, *RFC 2328: Open Shortest Path First Version 2 (OSPF V2)*, Apr. 1998.

[88] R. W. Floyd, "Algorithm 96: Ancestor," *Commun. ACM*, vol. 5, no. 6, pp. 344–345, Jun. 1962. [Online]. Available: http://doi.acm.org/10.1145/367766.368166

[89] K. Fall, "Network emulation in the vint/ns simulator," in *Computers and Communications, 1999. Proceedings. IEEE International Symposium on*, 1999, pp. 244 –250.

[90] A. Varga, "The omnet++ discrete event simulation system," *Proceedings of the European Simulation Multiconference (ESM'2001)*, June 2001.

[91] R. Chiocchetti, D. Rossi, and G. Rossini, "ccnsim: An highly scalable ccn simulator," in *Communications (ICC), 2013 IEEE International Conference on*, June 2013, pp. 2309–2314.

[92] A. Afanasyev, I. Moiseenko, L. Zhang *et al.*, "ndnsim: Ndn simulator for ns-3," *University of California, Los Angeles, Tech. Rep*, 2012.

[93] "Peersim: A peer-to-peer simulator, web site." [Online]. Available: http://peersim.sourceforge.net

[94] R. Buyya and M. Murshed, "Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002. [Online]. Available: http://dx.doi.org/10.1002/cpe.710

[95] I. Baumgart, B. Heep, and S. Krause, "Oversim: A flexible overlay network simulation framework," in *IEEE Global Internet Symposium, 2007*, may 2007, pp. 79 –84.

[96] W. Yang and N. Abu-Ghazaleh, "GPS: A general peer-to-peer simulator and its use for modeling bittorrent," in *Proceedings of MASCOTS*, 2005.

[97] R. M. Fujimoto, "Parallel discrete event simulation," in *Proceedings of the 21st conference on Winter simulation*, ser. WSC '89. New York, NY, USA: ACM, 1989, pp. 19–28. [Online]. Available: http://doi.acm.org/10.1145/76738.76741

[98] A. Rao, A. Legout, and W. Dabbous, "Can realistic bittorrent experiments be performed on clusters?" in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, aug. 2010, pp. 1 –10.

[99] Cisco Virtual Network, "http://www.cisco.com/web/cy/ solutions/sp/sp_strategy," 2009.

[100] T. Leighton, "Improving performance on the internet," *Commun. ACM*, vol. 52, no. 2, pp. 44–51, 2009.

[101] B. Ager, F. Schneider, J. Kim, and A. Feldmann, "Revisiting cacheability in times of user generated content," in *Proceedings of the 13th IEEE Global Internet Symposium*. IEEE, Mar. 2010.

[102] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *In Proceedings of the 11th SIGMETRICS*, New York, NY, USA, 2009.

[103] W. Wong, M. Giraldi, M. Magalhaes, and J. Kangasharju, "Content routers: Fetching data on network path," *IEEE International Conference on Communications (ICC)*, pp. 1 – 6, June 2011.

[104] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring isp topologies with rocketfuel," *Networking, IEEE/ACM Transactions on*, vol. 12, no. 1, pp. 2 – 16, feb. 2004.

[105] A. Anand, V. Sekar, and A. Akella, "Smartre: an architecture for coordinated network-wide redundancy elimination," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 87–98, 2009.

[106] Ager, B.; Schneider, F.; Juhoon Kim; Feldmann, A., "Revisiting cacheability in times of user generated content," *IEEE INFOCOM Conference*, pp. 1–6, March 2010.

[107] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '09. New York, NY, USA: ACM, 2009, pp. 37–48. [Online]. Available: http://doi.acm.org/10.1145/1555349.1555355

[108] Riverbed Wan Optimization, "http://www.riverbed.com/."

[109] Juniper WXC590 Application Acceleration Platform, "http://www.juniper.net/us/en/products-services/application-acceleration/wxc-series."

[110] Bluecoat Mach5, "http://www.bluecoat.com/products/mach5."

[111] A. Anand, V. Sekar, and A. Akella, "Smartre: an architecture for coordinated network-wide redundancy elimination," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 87–98. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592580

[112] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: the implications of universal redundant traffic elimination," in *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication.* New York, NY, USA: ACM, 2008, pp. 219–230.

[113] E. Zohar, I. Cidon, and O. O. Mokryn, "The power of prediction: cloud bandwidth and cost reduction," in *Proceedings of the ACM SIGCOMM 2011 conference on SIGCOMM*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 86–97. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018447

[114] D. Perino, M. Varvello, and K. P. N. Puttaswamy, "Icn-re: Redundancy elimination for information-centric networking," in *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*, ser. ICN '12. New York, NY, USA: ACM, 2012, pp. 91–96. [Online]. Available: http://doi.acm.org/10.1145/2342488.2342508

[115] W. Wong, M. Magalhães and J. Kangasharju, "Content Routers: Fetching Data on Network Path," *International Conference on Communications (ICC'11), Kyoto, Japan*, June 2011.

[116] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '00. New York, NY, USA: ACM, 2000, pp. 87–95. [Online]. Available: http://doi.acm.org/10.1145/347059.347408

[117] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "Endre: an end-system redundancy elimination service for enterprises," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, 2010.

[118] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *Proceedings of IEEE Infocom*, Anchorage, AK, Apr. 22–26, 2001.

[119] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content-centric networking," in *Proceedings of the 23rd International Teletraffic Congress*, ser. ITC '11. ITCP, 2011, pp. 111–118. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043468.2043487

[120] D. Wessels and K. Claffy, "RFC 2186: Internet Cache Protocol (ICP), version 2," Sep. 1997. [Online]. Available: www.rfc-editor.org/rfc/rfc2186.txt

[121] Paul, S. Yates, R. Raychaudhuri, D. Kurose, J., "The cache-and-forward network architecture for efficient mobile content delivery services in the future internet," *Innovations in NGN: Future Network and Services*, pp. 367–374, May 2008.

[122] P. Srebrny, T. Plagemann, V. Goebel, and A. Mauthe, "Cachecast: Eliminating redundant link traffic for single source multiple destination transfers," *In ICDCS'10*, pp. 209–220, 2010.

[123] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, "ROFL: Routing on Flat Labels," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 363–374, 2006.

[124] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. Thornton, D. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos *et al.*, "Named data networking (ndn) project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[125] Caida, "Greedy forwarding on the ndn testbed," August 2011. [Online]. Available: http://www.caida.org/research/routing/greedy_forwarding_ndn/

[126] D.-h. Kim, J.-h. Kim, Y.-s. Kim, H.-s. Yoon, and I. Yeom, "Mobility support in content centric networks," in *Proceedings of the second edition of the ICN workshop on Information-centric networking*, ser. ICN '12. New York, NY, USA: ACM, 2012, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/2342488.2342492

[127] A. Cvetkovski and M. Crovella, "Hyperbolic embedding and routing for dynamic graphs," in *INFOCOM 2009, IEEE*, april 2009, pp. 1647 –1655.

[128] R. Kleinberg, "Geographic routing using hyperbolic space," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, may 2007, pp. 1902 –1909.

[129] R. L. Rivest and B. Lampson, "Sdsi - a simple distributed security infrastructure," in *Technical Report*. MIT, 1996.

[130] A. Cvetkovski and M. Crovella, "Low-stretch greedy embedding heuristics," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, march 2012, pp. 232 –237.

[131] J. Herzen, C. Westphal, and P. Thiran, "Scalable routing easy as pie: A practical isometric embedding protocol," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, Oct 2011, pp. 49–58.

[132] A. Hofer, S. Roos, and T. Strufe, "Greedy embedding, routing and content addressing for darknets," in *Networked Systems (NetSys), 2013 Conference on*, March 2013, pp. 43–50.

[133] G. Rossini and D. Rossi, "Evaluating ccn multi-path interest forwarding strategies," *Computer Communications*, 2013.

[134] V. Sourlas, L. Gkatzikis, P. Flegkas, and L. Tassiulas, "Distributed cache management in information-centric networks," *Network and Service Management, IEEE Transactions on*, vol. PP, no. 99, pp. 1–14, 2013.

[135] S. Vanichpun and A. M. Makowski, "The output of a cache under the independent reference model: where did the locality

of reference go?" *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 295–306, Jun. 2004. [Online]. Available: http://doi.acm.org/10.1145/1012888.1005722

[136] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," in *Proceedings of the Re-Architecting the Internet Workshop*, ser. ReARCH '10. New York, NY, USA: ACM, 2010, pp. 5:1–5:6. [Online]. Available: http://doi.acm.org/10.1145/1921233.1921240

[137] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking*, ser. ICN '11. New York, NY, USA: ACM, 2011, pp. 44–49. [Online]. Available: http://doi.acm.org/10.1145/2018584.2018596

[138] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proceedings of IEEE INFOCOM*. IEEE, 2010, pp. 1–9.

[139] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. McGraw Hill, 2006.

[140] S. Choi, S. Cha, and C. Tappert, "A survey of binary similarity and distance measures," *Journal of Systemics, Cybernetics and Informatics*, vol. 8, no. 1, pp. 43–48, 2010.

[141] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, june 2008, pp. 229 –238.

[142] 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 on Internet measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 15–28. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298310

[143] U. Devi, R. Polavarapu, M. Chetlur, and S. Kalyanaraman, "On the partial caching of streaming video," in *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*, ser. IWQoS '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 4:1–4:9. [Online]. Available: http://dl.acm.org/citation.cfm?id=2330748.2330752

[144] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 1999, pp. 1310–1319 vol.3.

[145] S. Jin, A. Bestavros, and A. Iyengar, "Accelerating internet streaming media delivery using network-aware partial caching," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, 2002, pp. 153–160.

[146] J. Yu, C. T. Chou, X. Du, and T. Wang, "Internal popularity of streaming video and its implication on caching," in *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, vol. 1, 2006, pp. 6 pp.–.

[147] K.-L. Wu, P. Yu, and J. Wolf, "Segmentation of multimedia streams for proxy caching," *Multimedia, IEEE Transactions on*, vol. 6, no. 5, pp. 770–780, 2004.

[148] R. Rejaie and J. Kangasharju, "Mocha: a quality adaptive multimedia proxy cache for internet streaming," in *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, ser. NOSSDAV '01. New York, NY, USA: ACM, 2001, pp. 3–10. [Online]. Available: http://doi.acm.org/10.1145/378344.378345

[149] S.-H. Park, E.-J. Lim, and K.-D. Chung, "Popularity-based partial caching for vod systems using a proxy server," in *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, ser. IPDPS '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 115–. [Online]. Available: http://dl.acm.org/citation.cfm?id=645609.662482

[150] K.-L. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," in *Proceedings of the 10th international conference on World Wide Web*, ser. WWW '01. New York, NY, USA: ACM, 2001, pp. 36–44. [Online]. Available: http://doi.acm.org/10.1145/371920.371933

[151] V. Pacifici and G. Dán, "Selfish content replication on graphs," in *Proceedings of the 23rd International Teletraffic Congress*, ser. ITC '11. ITCP, 2011, pp. 119–126. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043468.2043488

[152] E. Rosensweig and J. Kurose, "Breadcrumbs: Efficient, best-effort content location in cache networks," in *INFOCOM 2009, IEEE*, April 2009, pp. 2631–2635.

[153] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale vod system," in *Proceedings of the 6th International Conference*, ser. Co-NEXT '10. New York, NY, USA: ACM, 2010, pp. 4:1–4:12. [Online]. Available: http://doi.acm.org/10.1145/1921168.1921174

[154] E. Rosensweig, D. Menasche, and J. Kurose, "On the steady-state of cache networks," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 863–871.

[155] E. Rosensweig and J. Kurose, "A network calculus for cache networks," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 85–89.

[156] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2001, pp. 1416–1424 vol.3.

[157] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–9.

[158] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and evaluation of ccn-caching trees," in *NETWORKING 2011*. Springer, 2011, pp. 78–91.

[159] S. Hasan, S. Gorinsky, C. Dovrolis, and R. K. Sitaraman, "Trade-offs in optimizing the cache deployments of cdns," in *INFOCOM, 2014 Proceedings IEEE*, 2014.

[160] N. Laoutaris, O. Telelis, V. Zissimopoulos, and I. Stavrakakis, "Distributed selfish replication," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 12, pp. 1401–1413, Dec 2006.

[161] B.-G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. H. Papadimitriou, and J. Kubiatowicz, "Selfish caching in distributed systems: a game-theoretic analysis," in *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, ser. PODC '04. New York, NY, USA: ACM, 2004, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/101⟩1767.1011771

[162] J. Kurose, "Information-centric networking: The evolution from circuits to packets to content," *Computer Networks*, vol. 66, no. 0, pp. 112 – 120, 2014, leonard Kleinrock Tribute Issue: A Collection of Papers by his Students. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614001455

[163] G. Carofiglio, G. Morabito, L. Muscariello, I. Solis, and M. Varvello, "From content delivery today to information centric networking," *Computer Networks*, vol. 57, no. 16, pp. 3116 – 3127, 2013, information Centric Networking. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128613002156

[164] A. E. Motter and Y.-C. Lai, "Cascade-based attacks on complex networks," *Physical Review E*, vol. 66, no. 6, p. 065102, 2002.

[165] P. Crucitti, V. Latora, and M. Marchiori, "Model for cascading failures in complex networks," *Physical Review E*, vol. 69, no. 4, p. 045104, 2004.

[166] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: A survey," in *Approximation Algorithms for NP-hard Problems*, D. S. Hochbaum, Ed. Boston, MA, USA: PWS Publishing Co., 1997, pp. 46–93. [Online]. Available: http://dl.acm.org/citation.cfm?id=241938.241940

[167] J. Nash, John F., "The bargaining problem," *Econometrica*, vol. 18, no. 2, pp. pp. 155–162, 1950. [Online]. Available: http://www.jstor.org/stable/1907266

[168] A. Muthoo, *Bargaining theory with applications*. Cambridge University Press, 1999.

[169] D. Niu and B. Li, "An efficient distributed algorithm for resource allocation in large-scale coupled systems," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 1501–1509.

[170] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2009.

[171] S. Bornholdt, H. G. Schuster, and J. Wiley, *Handbook of graphs and networks*. Wiley Online Library, 2003, vol. 2.

[172] F. P. Kelly, A. K. Maulloo, and D. K. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research society*, vol. 49, no. 3, pp. 237–252, 1998.

[173] H. Boche and M. Schubert, "Nash bargaining and proportional fairness for wireless systems," *Networking, IEEE/ACM Transactions on*, vol. 17, no. 5, pp. 1453–1466, 2009.

[174] K.-W. Hwang, D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, V. Misra, K. K. Ramakrishnan, and D. F. Swayne, "Leveraging video viewing patterns for optimal content placement," in *IFIP NETWORKING 2012*. Springer, 2012, pp. 44–58.

[175] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.

[176] V. Pacifici, F. Lehrieder, and G. Dan, "Cache capacity allocation for bittorrent-like systems to minimize inter-isp traffic," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 1512–1520.

[177] S. Saha, A. Lukyanenko, and A. Yla-Jaaski, "Cooperative caching through routing control in information-centric networks," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 100–104.

[178] M. Amadeo, C. Campolo, and A. Molinaro, "Multi-source data retrieval in iot via named data networking," in *Proceedings of the 1st International Conference on Information-centric Networking*, ser. ICN '14. New York, NY, USA: ACM, 2014, pp. 67–76. [Online]. Available: http://doi.acm.org/10.1145/2660129.2660148

[179] G. Rossini and D. Rossi, "Coupling caching and forwarding: Benefits, analysis, and implementation," in *Proceedings of the 1st International Conference on Information-centric Networking*, ser. ICN '14. New York, NY, USA: ACM, 2014, pp. 127–136. [Online]. Available: http://doi.acm.org/10.1145/2660129.2660153

[180] V. Pacifici and G. Dan, "Content-peering dynamics of autonomous caches in a content-centric network," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 1079–1087.

TIETOJENKÄSITTELYTIETEEN LAITOS
PL 68 (Gustaf Hällströmin katu 2 b)
00014 Helsingin yliopisto

DEPARTMENT OF COMPUTER SCIENCE
P.O. Box 68 (Gustaf Hällströmin katu 2 b)
FI-00014 University of Helsinki, FINLAND

JULKAISUSARJA **A**

SERIES OF PUBLICATIONS **A**

Reports are available on the e-thesis site of the University of Helsinki.

A-2010-1 M. Lukk: Construction of a global map of human gene expression - the process, tools and analysis. 120 pp. (Ph.D. Thesis)

A-2010-2 W. Hämäläinen: Efficient search for statistically significant dependency rules in binary data. 163 pp. (Ph.D. Thesis)

A-2010-3 J. Kollin: Computational Methods for Detecting Large-Scale Chromosome Rearrangements in SNP Data. 197 pp. (Ph.D. Thesis)

A-2010-4 E. Pitkänen: Computational Methods for Reconstruction and Analysis of Genome-Scale Metabolic Networks. 115+88 pp. (Ph.D. Thesis)

A-2010-5 A. Lukyanenko: Multi-User Resource-Sharing Problem for the Internet. 168 pp. (Ph.D. Thesis)

A-2010-6 L. Daniel: Cross-layer Assisted TCP Algorithms for Vertical Handoff. 84+72 pp. (Ph.D. Thesis)

A-2011-1 A. Tripathi: Data Fusion and Matching by Maximizing Statistical Dependencies. 89+109 pp. (Ph.D. Thesis)

A-2011-2 E. Junttila: Patterns in Permuted Binary Matrices. 155 pp. (Ph.D. Thesis)

A-2011-3 P. Hintsanen: Simulation and Graph Mining Tools for Improving Gene Mapping Efficiency. 136 pp. (Ph.D. Thesis)

A-2011-4 M. Ikonen: Lean Thinking in Software Development: Impacts of Kanban on Projects. 104+90 pp. (Ph.D. Thesis)

A-2012-1 P. Parviainen: Algorithms for Exact Structure Discovery in Bayesian Networks. 132 pp. (Ph.D. Thesis)

A-2012-2 J. Wessman: Mixture Model Clustering in the Analysis of Complex Diseases. 118 pp. (Ph.D. Thesis)

A-2012-3 P. Pöyhönen: Access Selection Methods in Cooperative Multi-operator Environments to Improve End-user and Operator Satisfaction. 211 pp. (Ph.D. Thesis)

A-2012-4 S. Ruohomaa: The Effect of Reputation on Trust Decisions in Inter-enterprise Collaborations. 214+44 pp. (Ph.D. Thesis)

A-2012-5 J. Sirén: Compressed Full-Text Indexes for Highly Repetitive Collections. 97+63 pp. (Ph.D. Thesis)

A-2012-6 F. Zhou: Methods for Network Abstraction. 48+71 pp. (Ph.D. Thesis)

A-2012-7 N. Välimäki: Applications of Compressed Data Structures on Sequences and Structured Data. 73+94 pp. (Ph.D. Thesis)

A-2012-8 S. Varjonen: Secure Connectivity With Persistent Identities. 139 pp. (Ph.D. Thesis)

A-2012-9 M. Heinonen: Computational Methods for Small Molecules. 110+68 pp. (Ph.D. Thesis)

A-2013-1 M. Timonen: Term Weighting in Short Documents for Document Categorization, Keyword Extraction and Query Expansion. 53+62 pp. (Ph.D. Thesis)

A-2013-2 H. Wettig: Probabilistic, Information-Theoretic Models for Etymological Alignment. 130+62 pp. (Ph.D. Thesis)

A-2013-3 T. Ruokolainen: A Model-Driven Approach to Service Ecosystem Engineering. 232 pp. (Ph.D. Thesis)

A-2013-4 A. Hyttinen: Discovering Causal Relations in the Presence of Latent Confounders. 107+138 pp. (Ph.D. Thesis)

A-2013-5 S. Eloranta: Dynamic Aspects of Knowledge Bases. 123 pp. (Ph.D. Thesis)

A-2013-6 M. Apiola: Creativity-Supporting Learning Environments: Two Case Studies on Teaching Programming. 62+83 pp. (Ph.D. Thesis)

A-2013-7 T. Polishchuk: Enabling Multipath and Multicast Data Transmission in Legacy and Future Interenet. 72+51 pp. (Ph.D. Thesis)

A-2013-8 P. Luosto: Normalized Maximum Likelihood Methods for Clustering and Density Estimation. 67+67 pp. (Ph.D. Thesis)

A-2013-9 L. Eronen: Computational Methods for Augmenting Association-based Gene Mapping. 84+93 pp. (Ph.D. Thesis)

A-2013-10 D. Entner: Causal Structure Learning and Effect Identification in Linear Non-Gaussian Models and Beyond. 79+113 pp. (Ph.D. Thesis)

A-2013-11 E. Galbrun: Methods for Redescription Mining. 72+77 pp. (Ph.D. Thesis)

A-2013-12 M. Pervilä: Data Center Energy Retrofits. 52+46 pp. (Ph.D. Thesis)

A-2013-13 P. Pohjalainen: Self-Organizing Software Architectures. 114+71 pp. (Ph.D. Thesis)

A-2014-1 J. Korhonen: Graph and Hypergraph Decompositions for Exact Algorithms. 62+66 pp. (Ph.D. Thesis)

A-2014-2 J. Paalasmaa: Monitoring Sleep with Force Sensor Measurement. 59+47 pp. (Ph.D. Thesis)

A-2014-3 L. Langohr: Methods for Finding Interesting Nodes in Weighted Graphs. 70+54 pp. (Ph.D. Thesis)

A-2014-4 S. Bhattacharya: Continuous Context Inference on Mobile Platforms. 94+67 pp. (Ph.D. Thesis)

A-2014-5 E. Lagerspetz: Collaborative Mobile Energy Awareness. 60+46 pp. (Ph.D. Thesis)