# Cooperative Caching Techniques for Named Data Networks

**Ph.D Thesis**

by

**Pankaj Chaudhary**


**DEPARTMENT OF COMPUTER SCIENCE**

**AND ENGINEERING**

**INDIAN INSTITUTE OF TECHNOLOGY**

**INDORE**

**May 2026**

# Cooperative Caching Techniques for Named Data Networks

**A THESIS**

*Submitted in partial fulfillment of the*

*requirements for the award of the degree*

*of*

**DOCTOR OF PHILOSOPHY**

by

**Pankaj Chaudhary**

**2001201004**


**DEPARTMENT OF COMPUTER SCIENCE**

**AND ENGINEERING**

**INDIAN INSTITUTE OF TECHNOLOGY**

**INDORE**

**May 2026**

# Cooperative Caching Techniques for Named Data Networks

By

Pankaj Chaudhary

A Thesis Submitted to

Indian Institute of Technology Indore

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Approved:

Prof. Neminath Hubballi

Thesis Advisor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2026

# INDIAN INSTITUTE OF TECHNOLOGY INDORE

## CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Cooperative Caching Techniques for Named Data Networks** in the partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy** and submitted in the **Department of Computer Science and Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the period from December 2020 to December 2025 under the supervision of Prof. Neminath Hubballi, Indian Institute of Technology Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

26/05/2026

Signature of the Student with Date

**(Pankaj Chaudhary)**

---

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

 26/05/2026

Signature of Thesis Supervisor with Date

**(Prof. Neminath Hubballi)**

---

**Pankaj Chaudhary** has successfully given his Ph.D. Oral Examination held on **25 May 2026**

 26/05/2026

Signature of Thesis Supervisor with Date

**(Prof. Neminath Hubballi)**

# ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to everyone who supported and encouraged me throughout my research journey. Their guidance, motivation, and kindness made this experience meaningful, rewarding, and truly enjoyable.

First and foremost, I sincerely thank my supervisor, **Prof. Neminath Hubballi**, for his continuous support and guidance throughout my research. His mentorship has shaped my approach and was instrumental in the successful completion of this work. I deeply appreciate the time, patience, and trust he has placed in me.

I am also grateful to our collaborator, **Dr. Sameer G. Kulkarni** (IIT Gandhinagar), for his insightful discussions and valuable feedback, which played a significant role in shaping the direction of my research.

I sincerely thank my research progress committee members, **Prof. Gourinath Banda** and **Prof. Niraj Kumar Shukla**, for their continuous support and valuable feedback over the years, which greatly improved my work at every stage.

I extend my sincere appreciation to **Prof. Suhas S. Joshi**, Director of IIT Indore, for providing me the opportunity to pursue my research at this esteemed institute. I am also grateful to the Head, faculty members, and staff of the CSE Department for their continuous academic and administrative support.

I gratefully acknowledge the financial support from the UGC, India, throughout my research. I am grateful to the NDN research community across the globe for their contributions, shared resources, and invaluable publications that inspired and guided my work. I am grateful to my colleagues, lab mates, and friends for their cooperation, encouragement, and companionship, which made this journey enjoyable and fulfilling.

Finally, I am deeply thankful to my parents for their unconditional love and support, which have been my greatest motivation. I am especially grateful to my wife, Neha, for her encouragement, patience, and understanding throughout the journey. I also extend thanks to my siblings and family members for their love and support.

To everyone who supported me on this journey, I am truly grateful.

***Pankaj Chaudhary***

***Dedicated to My Family***

*who showed unwavering patience, love, and faith through every step of this journey.*

# Publications

## Publications Related to the Thesis

## Published/Accepted

## Journals

**J1.** **Pankaj Chaudhary**, Neminath Hubballi, and Sameer G. Kulkarni, *"eNCache: Improving Content Delivery with Cooperative Caching in Named Data Networking"*, Computer Networks, vol. 237, pp. 1–13, 2023. doi:10.1016/j.comnet.2023.110104

**J2.** **Pankaj Chaudhary** and Neminath Hubballi, *"PeNCache: Popularity based cooperative caching in Named Data Networks"*, Computer Networks, vol. 257, pp. 1–13, 2025. doi:10.1016/j.comnet.2024.110995

**J3.** **Pankaj Chaudhary**, Neminath Hubballi, and Sameer G. Kulkarni, *"DPCCache: Demand Driven Content Popularity based Cooperative Caching in Named Data Networks"*, IEEE China Communications, pp. 1–22, 2025. *(To appear)*

## Conferences

**C1.** **Pankaj Chaudhary**, Neminath Hubballi, and Sameer G. Kulkarni, *"NCache: Neighborhood Cooperative Caching in Named Data Networking"*, Proceedings of the 5th International Conference on Hot Information-Centric Networking (HotICN), Guangzhou, China, November 24–26, 2022, pp. 36–41, IEEE. doi:10.1109/HotICN57539.2022.10036203

**C2.** Neminath Hubballi, **Pankaj Chaudhary**, and Sameer G. Kulkarni, *"PePC: Popularity Based Early Predictive Caching in Named Data Networks"*, Proceedings of the IEEE 21st Consumer Communications and Networking Conference (CCNC), Las Vegas, USA, January 06–09, 2024, pp. 478–483, IEEE. doi:10.1109/CCNC51664.2024.10454826

**C3.** Neminath Hubballi and **Pankaj Chaudhary**, *"CPCache: Cooperative Popularity based Caching for Named Data Networks"*, Proceedings of the 38th International Conference on Information Networking (ICOIN), Ho Chi Minh City, Vietnam, January 17–19, 2024, pp. 379–384, IEEE. doi:10.1109/ICOIN59985.2024.10572135

**C4.** **Pankaj Chaudhary** and Neminath Hubballi, *"EeNCache: Neighborhood Cooperative Content Caching and Delivery in Named Data Networks"*, Proceedings of the 12th International Conference on Computing, Networking and Communications (ICNC), Honolulu, Hawaii, USA, February 17–20, 2025, pp. 557–561, IEEE. doi:10.1109/ICNC64010.2025.10993651

**C5.** **Pankaj Chaudhary** and Neminath Hubballi, *"RepCache: Content Producer Reputation based Caching in Named Data Networks"*, Proceedings of the IEEE 23rd Consumer Communications and Networking Conference (CCNC), Las Vegas, USA, January 09–12, 2026, pp. 1–4, IEEE. doi:10.1109/CCNC65079.2026.11366409

## Unpublished/Under Review

**U1.** **Pankaj Chaudhary**, Neminath Hubballi, and Sameer G. Kulkarni, *"CPePC: Cooperative and Predictive Popularity based Caching for Named Data Networks"*, arXiv preprint arXiv:2512.24073 [cs.NI], 2025. doi:10.48550/arXiv.2512.24073

**U2.** **Pankaj Chaudhary**, Neminath Hubballi, and Sameer G. Kulkarni, *"Content Caching Methods in Named Data Networks"*, arXiv preprint arXiv:2605.13104 [cs.NI], 2026. doi:10.48550/arXiv.2605.13104

**U3.** **Pankaj Chaudhary**, Neminath Hubballi, and Sameer G. Kulkarni, *"Routing in Named Data Networks: A Survey"*. (*Under review, IEEE China Communications*)

## Other Publications

**O1.** **Pankaj Chaudhary**, Aditi Aralkar, Neminath Hubballi, Vinduja T, Paromita Choudhury, and Manjesh Kumar Hanawal, *"Website Fingerprinting Attacks and Defense Techniques: A Survey"*, ACM Computing Surveys, pp. 1–37, 2026. *(To appear)*

**O2.** Aditi Aralkar, **Pankaj Chaudhary**, Neminath Hubballi, and Manjesh Kumar Hanawal, *"Rethinking Website Fingerprinting: A Critical Study"*, 18th International Conference on COMmunication Systems and NETworks (COMSNETS), Bengaluru, India, January 06–10, 2026, pp. 603–608, IEEE. doi:10.1109/COMSNETS67989.2026.11418290

# ABSTRACT

Information Centric Networking (ICN) is a new paradigm of network architecture that makes content a first-class citizen. Named Data Networking (NDN) is a specific implementation of ICN, built on a content-centric communication model to address challenges posed by the current Internet architecture. Unlike the present TCP/IP model, in NDN, communication is no longer based on the location of content; instead, it uses the name of the content itself to fetch the desired data. It is a promising architecture for the future Internet due to its wide range of features, such as content caching in network routers, mobility support, efficient multicast forwarding, scalability, and data authenticity and integrity through per-packet signatures. The in-network caching feature of NDN has drawn significant interest from researchers because it improves content accessibility by storing data closer to end users. Since NDN routers have limited cache space, content is stored temporarily, and when the cache is full, newly arriving content replaces older content. Therefore, designing an effective caching mechanism is essential to enhance user experience and reduce the load on the server. One way to better utilize the available cache space is to bring cooperation among the nodes.

In this thesis, we design several cooperative caching techniques to make use of the content available in nearby routers, aiming to improve the cache hit ratio and reduce content access time. Our first contribution in this thesis is `eNCache`, which is a cooperative content lookup and caching technique. `eNCache` is designed to efficiently search for content available in the caches of nearby routers outside the transmission path. It also does cooperative decisions while caching content. As our second contribution, we designed `PeNCache`, a technique that incorporates content popularity into the caching decisions. To make cooperation easy, in our next contribution, we design `DPCCache`, which divides the network topology into multiple communities or clusters and elects a leader node in each community through which request forwarding, caching, and popularity estimation decisions are made. Our final contribution introduces a community-centric technique, `CPePC`, which considers both content popularity and router occupancy when caching content to better utilize cache space. To assess the effectiveness of our proposed caching techniques, we conducted simulations using the Icarus simulator. We compared them with state-of-the-art methods over large scale Internet topologies. The simulation results demonstrate that our cooperative techniques perform well measured in terms of *cache hit ratio*, *access latency*, *cache diversity*, and *hit distance*.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

The Internet has become an integral part of our daily life. It is used for various activities such as email communication, education, e-commerce, banking, social media, entertainment, and many more. The current Internet architecture, the way it works today, was designed almost 60 years ago with a few use cases in mind and heavily influenced by the way computers worked in that period. In its early days, the network/Internet was used mainly for email communication and exchanging files between researchers and institutions. However, over the last five decades, the Internet has witnessed several new applications, and this, coupled with developments in data transmission technologies, has led to widespread changes in usage patterns. Several reports [1, 2] highlight that the majority of today's Internet traffic is dominated by rich multimedia content, driven by the rise of video streaming services. The current Internet architecture is based on the Transmission Control Protocol/Internet Protocol (TCP/IP) [26, 87] stack, which uses IP addresses to establish communication between devices. This location-centric Internet is facing significant challenges in handling the continuously rising volume of content-centric Internet traffic. To improve content distribution on today's Internet, various overlay solutions [137] have been proposed by researchers, such as Content Delivery Networks (CDNs) [81], Peer-to-Peer Networking (P2P) [11], IP Anycast [15], IP Multicast [41], Transparent Caching [36], *etc.* However, these solutions have their

own limitations, sparking a demand for a new Internet architecture that can address the challenges of the IP host-centric model.

A major approach that emerged is the departure from the host-centric model in the form of Information-Centric Networking (ICN) [163, 169] for better content distribution. ICN is based on a content-centric communication model. In order to realize this communication and content delivery, different network architectures were proposed. Some noted ICN architectures include Data-Oriented Network Architecture (DONA) [78], Content-Centric Networking (CCN) [70], Content-Centric Inter-Networking (CONET) [42], publish/subscribe paradigm [49], Network of Information (NetInf) [37], and Named Data Networking (NDN) [180]. Among these ICN architectures, NDN is a well-established and promising architecture that supports name-based routing and caching operations [144]. In recent years, NDN has remained a popular ICN architecture among researchers due to its simple architectural design and consistent support from the NDN community. The idea of NDN is to focus on getting the content itself, instead of who provides it and from where it is served. NDN architecture inherently supports network-level caching, multicast forwarding, and content-level security, where each content is signed by the provider and verified by the receiver. These features make NDN a promising solution for future Internet architecture to efficiently handle content distribution.

## 1.1 NDN Architecture Overview

Communication between the client and server in NDN differs from that in IP networks. The communication model of NDN is shown in Figure 1.1, depicting how messages are exchanged between entities in the NDN network. NDN uses different packet types for requests and responses. A request is made using the *Interest packet*, which is generated by consumers (end users) to retrieve the content, and the provider, having the matching requested content, responds with the *Data packet*. Note that the content provider can be either the original

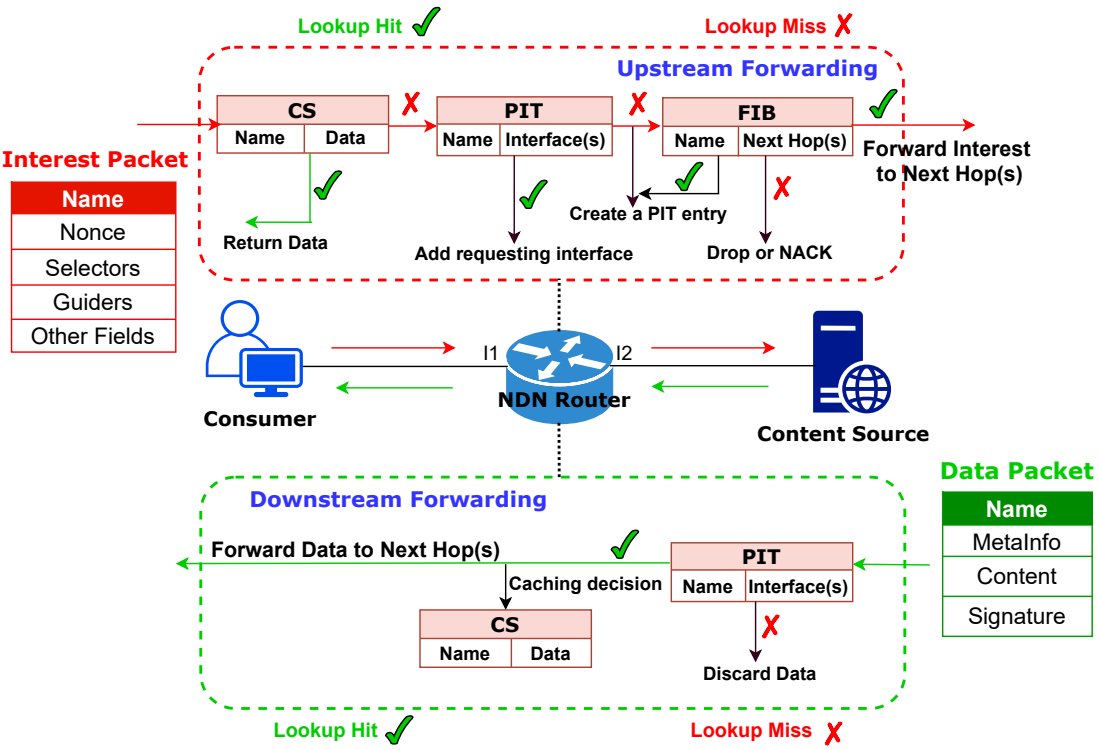



Figure 1.1: NDN Forwarding Daemon (NFD) Data Structures for Interest and Data Forwarding.

content source or any router in the network, as NDN supports network-level caching. The Interest packet is forwarded towards the provider, a process called upstream forwarding, while the Data packet is forwarded back towards the consumer, known as downstream forwarding. Both packets are forwarded in upward and downward directions using the hierarchical naming, for example, */ndn/videos/Name.mp4*. These two packets move from one node to another in the network using the three primary data structures, as shown in Figure 1.1. Each NDN router maintains the following three data structures for handling packets:

*(i) Content Store (CS):* This data structure is used for temporarily caching the content.

*(ii) Pending Interest Table (PIT):* PIT keeps track of requests sent to upstream routers. It stores all incoming interfaces from where the request came until the content arrives or the entry times out. Due to this data structure, the NDN forwarding plane becomes stateful

and efficiently supports multicast forwarding.

*(iii) Forwarding Information Base (FIB):* This table stores routing information to forward requests to the next hop(s). The role of the FIB is similar to that in IP networks, but instead of IP addresses, it stores name prefixes, with multiple outgoing interfaces for each prefix.

An example here will help better understand the NDN packet forwarding operation. Let us consider the topology shown in Figure 1.1. When an NDN router receives an Interest packet for content `Name` from interface *I1*, it first checks its CS. If a match is found, the content is returned to interface *I1*. If no match is found, it checks PIT. If an entry for `Name` already exists in the PIT, it means the router has already received an Interest for the same content. In that case, the router adds interface *I1* to the existing PIT entry for `Name` and drops the Interest. If there is no PIT entry for `Name`, the router consults the FIB to forward the request to the next hop(s) and creates a new PIT entry for `Name`. This process is repeated at each hop until the content is found.

In the reverse direction, when an NDN router receives a Data packet from interface *I2*, it first checks the PIT for a matching entry. If a match is found, the router forwards the Data packet to all interfaces listed in the PIT for content `Name`, stores the content in its CS (according to caching policy), and removes the PIT entry. If no PIT entry is found, the router discards the Data packet, as the request has either already been satisfied or the PIT entry has expired.

## 1.2 Motivation for the Work

In-network caching is one of the most remarkable characteristics of the NDN architecture that significantly enhances content delivery performance. Routers along the content delivery path (*i.e.,* between the content requester and the provider) can store content to satisfy future requests. This approach not only speeds up content access for users but also reduces the load on the original content source. Caching is natively supported in NDN, where all routers

along the downstream path cache the content. For example, in Figure 1.2, content `Name` is served by the original content source in response to a request from Consumer1. The Data packet follows the reverse path of the Interest packet to deliver the content back to Consumer1. In this case, routers $R_4, R_3, R_2,$ and $R_1$ forward the content to Consumer1, and while forwarding, each router temporarily caches content `Name` in its CS to enable faster responses to future requests for `Name`.

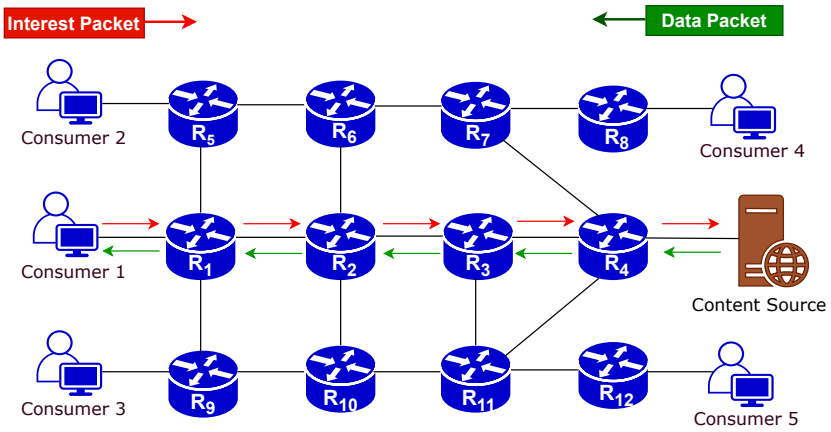



Figure 1.2: Network Topology.

However, routers have limited cache storage, and with the growing demand for multimedia content, caching all passing content at every router leads to poor cache space utilization, which degrades the performance of NDN. Many caching techniques [118, 131, 155] have been designed to address the limitations of the NDN default caching approach. Several researchers [10, 19, 111] have suggested incorporating content popularity into caching decisions to make more efficient use of cache space.

Although existing caching approaches improve cache utilization by considering factors such as content popularity, limited or non-cooperative caching decisions [7, 19, 60, 104, 118] increase the likelihood of redundant content placement across neighboring routers. Furthermore, excessive replication of highly popular content without proper coordination reduces network-wide cache diversity, where multiple routers store identical content instead of main-

taining a broader range of distinct content objects. Cache diversity [19, 129] represents the fraction of distinct content objects cached across the network relative to the total cached content. Reduced network-wide cache diversity limits the effective utilization of scarce cache resources and decreases the chances of satisfying diverse future content requests from nearby caches.

Our comprehensive analysis revealed several shortcomings in existing NDN caching techniques, indicating considerable scope for improving cache resource utilization and content retrieval performance. The following limitations are associated with existing caching techniques:

(i) Many existing on-path caching algorithms [10, 59, 93, 100, 118] overlook the content available in the caches of nearby off-path routers during request forwarding towards the original content source. This on-path approach reduces the performance of the caching system. For example, in Figure 1.2, content `Name` is published by the content source, and a copy is also stored in the cache of router $R_5$. When Consumer1 sends a request for `Name`, the traditional best-path forwarding strategy of NDN forwards the request through routers $R_1, R_2, R_3,$ and $R_4$, eventually retrieving the content from the original source. This approach takes 10 hops to retrieve `Name`. On the other hand, if the strategy also looks for content available in the cache of routers outside the best path between the consumer and the original source, `Name` can be retrieved from the off-path router $R_5$, requiring only 4 hops.

(ii) Existing off-path techniques [13, 61, 62, 106, 112] utilize content available outside the transmission path to improve the cache hit ratio and content delivery time. However, these approaches require each router to maintain information about the availability of content in neighboring nodes. The drawback of this approach is twofold. First, each router needs to maintain an additional data structure, which increases the storage complexity. Second, due to limited storage, the cache states of routers change frequently, so the information available in the table from previous exchanges becomes stale.

(iii) Many existing popularity-based caching strategies [7, 19, 56, 65, 191] only consider

local request patterns at individual routers, overlooking the global popularity of content. Estimating local popularity is easier compared to estimating global popularity [50, 90], but caching decisions based solely on local popularity do not provide benefits to many consumers.

Taking motivation from these limitations, we design new cooperative caching techniques to further improve caching performance in NDN. Cooperation among routers can improve cache utilization by maximizing cache hit ratio while reducing the chances of storing the same content redundantly at multiple locations in the network. However, another important aspect that must be considered in cooperative caching design is the enhancement of overall user experience in the NDN network, where users should receive the requested content with minimum response time from nearby caches. A higher cache hit ratio does not always guarantee lower content retrieval time, as the requested content may be cached at routers located far from consumers or additional delay may be introduced due to cooperation overhead. Such delays can adversely affect user experience in the NDN network. Therefore, along with improving cache hit ratio and cache diversity, cooperative caching mechanisms should also minimize content retrieval time to achieve efficient and user-centric content delivery in NDN.

## 1.3 Research Questions

Designing the new caching technique for NDN requires addressing the following questions, which directly impact the performance of any caching system.

(i) What content should be cached, and when should it be cached in order to improve the cache hit ratio and content diversity?

(ii) Where should the content be cached within the network in order to reduce retrieval time and redundancy?

(iii) Which content should be replaced when the cache becomes full?

(iv) To what extent does caching popular content improve content availability in the network?

(v) How can content discovery and cache management operations be simplified?

(vi) Can retrieving content from an off-path neighbor be more beneficial than fetching it from any on-path router or the original content source?

(vii) How can cooperation be established in the network while keeping signaling overhead minimal?

(viii) How can bandwidth usage and load on the original content source be minimized?

## 1.4 Objectives for the Work

The purpose of our thesis is to design new caching techniques for the NDN architecture to improve content delivery performance and user experience. To achieve this, we identify the following relevant objectives:

1. Maximize the cache hit ratio.
2. Minimize content retrieval time.
3. Improve cache diversity.

To achieve these objectives, we propose the following solutions:

(i) Cooperative content searching and caching techniques to enhance content retrieval and reduce redundancy.

(ii) Hybrid cooperative approaches that take advantage of content available in both on-path and off-path routers.

(iii) Popularity-aware content placement techniques that leverage local and global content popularity to optimize the use of limited router cache resources.

## 1.5 Contributions of the Thesis

In this thesis, we describe several cooperative caching techniques for NDN. The main contributions of the thesis are outlined below, with each one corresponding to a specific chapter.

***Contribution 1:*** In our first contribution, we design `eNCache`, a lightweight cooperative technique to address the performance degradation of caching systems caused by the default on-path request routing in NDN. `eNCache` explores *N-hop* off-path routers while forwarding content requests towards the potential provider. It simultaneously explores the next-hop on-path router and *N-hop* off-path routers to speed up content retrieval. Any nearby on-path or off-path node that has the requested content can serve it back to the consumer. To achieve this, we modified the native NDN Interest packet structure by adding additional fields that allow control over forwarding requests to on-path and off-path routers. `eNCache` improves the cache hit ratio, reduces content access time, and lowers the load on the origin content source by fetching content from nearby off-path neighbors. `eNCache` also reduces the memory burden on routers by eliminating the need to maintain cache information of neighboring nodes. In the reverse direction, routers also make caching decisions cooperatively to avoid content redundancy in the neighborhood. Packet forwarding in `eNCache` is not different from the symmetric forwarding model of NDN, where both requests and responses follow the same path. However, unlike traditional request forwarding, our approach also explores content available outside the path to respond more quickly to the consumer.

***Contribution 2:*** We improve on the `eNCache` work by incorporating content popularity into caching decisions. As not all content is frequently requested by consumers, caching rarely requested content may occupy cache space unnecessarily, which can degrade performance. We design `PeNCache`, which performs cooperative decision-making for content searching and caching, and improves cache utilization by prioritizing the caching of popular content. Similar to `eNCache`, `PeNCache` also explores both on-path and off-path neighbors simultaneously to reduce content retrieval time. `PeNCache` considers both local and global content popularity to distinguish between popular and unpopular content. For local popularity, each router maintains a table to track passing content requests. To determine global popularity, we identify a few nodes in the network based on high degree centrality, which are responsible for tracking the global content request counts. In order to support better

cooperative decision-making and reduce communication overhead, we add a few new fields to the native NDN packets.

***Contribution 3:*** A cooperative approach for content searching and caching helps make better use of the limited capacity of routers in the NDN network. However, establishing cooperation becomes challenging in large network topologies. When each node communicates directly with others, it leads to a significant increase in message exchanges across the network. Another challenge associated with this type of cooperation is estimating global content popularity. Since routers are far apart in a large topology, exchanging local information with other routers to determine the global view of content requests becomes difficult. Therefore, to overcome these challenges, our third contribution proposes `DPCCache`, a semi-decentralized approach that divides the network topology into smaller communities to simplify cooperation. In each community, a router with higher degree centrality is elected as the leader node, which is responsible for making decisions on routing, caching, and popularity estimation. Content searching and caching are handled with the help of the leader node, which reduces the burden on other routers. Each router communicates only with the leader instead of sending messages to all its neighbors. `DPCCache` ensures no redundancy within the community by assigning caching decisions to the leader, who checks that the content is not already available locally before caching it. As all content is unique within the community, `DPCCache` re-caches content evicted from one router into another router with available space to enhance content availability. Popularity estimation in `DPCCache` is demand-driven, where popularity is measured only for the requested content. This approach allows it to quickly capture sudden changes in demand, *e.g.,* viral news in a specific region.

***Contribution 4:*** Building on previous work on community division, we design `CPePC` to improve cache utilization and reduce communication overhead. Similar to `DPCCache`, `CPePC` partitions the topology into several communities and elects a leader node for each community. After partitioning, the router with the highest betweenness centrality is designated as the community leader. The leader node performs content searching, cache management, and

popularity estimation. Furthermore, we modified the native NDN packet structure to make content searching and caching decisions easier within the community. `CPePC` considers both local and global request counts to periodically estimate the threshold that distinguishes between popular and unpopular content. For caching decisions, `CPePC` dynamically estimates router cache occupancy to better utilize the limited cache space. We divide the caching decision into three scenarios based on the minimum and maximum occupancy thresholds. If the cache occupancy is below the minimum threshold, the router has enough space to cache any content. If the occupancy is above the maximum threshold, the cache space is critical, so the router caches only popular content. When the occupancy is between the minimum and maximum thresholds, we design a predictive model to determine which content to allow into the cache to maximize benefit.

Our proposed caching algorithms presented in this thesis are evaluated using the Icarus [135] simulator, which is a widely used simulator for evaluating the performance of caching techniques in ICN/NDN. We used large-scale RocketFuel ISP topologies [146, 147] to test the effectiveness of the proposed methods. The performance comparison of the proposed caching techniques is done against state-of-the-art caching techniques using well-known evaluation metrics such as *cache hit ratio, content access time, hit distance*, and *cache diversity.* To verify the performance of our proposed caching techniques against state-of-the-art methods, we conducted simulations by tuning various simulator parameters in order to closely resemble real-world conditions. From the simulation outcomes, we observed that our proposed cooperative caching techniques outperform state-of-the-art methods under various simulation conditions.

## 1.6 Thesis Organization

The remainder of the thesis is organized as follows:

**Chapter 2**: This chapter presents the workings of various caching strategies described in

the NDN literature.

**Chapter 3**: In this chapter, we describe the working of `eNCache`, which enables lightweight cooperation among neighborhoods to improve the cache hit ratio and speed up content retrieval.

**Chapter 4**: In this chapter, we present the working of `PeNCache`, which establishes cooperation among neighborhoods and also considers content popularity to better utilize the cache storage of routers.

**Chapter 5**: This chapter introduces `DPCCache`, which divides a large network topology into multiple communities to simplify cooperation and improve the cache hit ratio and content diversity.

**Chapter 6**: This chapter presents `CPePC`, which determines caching decisions by considering both the router's available storage and the popularity of content to enhance the cache hit ratio.

**Chapter 7**: This chapter summarizes the thesis contributions and suggests potential directions for future work in this area.

# Chapter 2

# Literature Survey

Caching has become a key component of computing and networking systems, helping reduce the communication and computational costs of exchanging data [162, 177]. Caching is used to temporarily store copies of content objects closer to the user, which improves the user experience and reduces network traffic. Storing copies of content within the network helps reduce the load on the server. For example, during peak hours of ticket booking, this prevents the server from becoming overloaded and improves response time. As network traffic increased with the growing size of the Internet, researchers have developed web caching systems [16, 28, 39, 40, 154, 164] that cache web objects to enhance the performance of web applications. Caching is used in various applications such as Distributed File Systems (DFS) [20, 58], File Transfer Protocol (FTP) [38], Domain Name Systems (DNS) [73], Content Distribution Networks (CDN) [22], and Peer-to-Peer (P2P) systems [148]. Caching in DNS is very beneficial, as DNS performs hierarchical queries to resolve domain names into network addresses through Root DNS servers, TLD DNS servers, and Authoritative DNS servers. Caching DNS answers in local resolvers reduces the time needed to map host names to network addresses

With a similar motivation, the NDN architecture adopted the caching at the network layer [70, 180] to store content that can satisfy future requests directly from the network

cache. In the NDN architecture, in-network caching is an important feature that enhances its effectiveness by strategically storing content in the network. Just like traditional web caching, caching techniques in the NDN architecture also perform two essential operations: i) the Content Placement Decision and ii) the Content Replacement Decision. Content placement decisions involve routers making informed choices about whether to cache a specific piece of content and where to cache it. On the other hand, the content replacement decision is the subsequent step, where the routers determine which content to evict from their cache to make room for new content when necessary. This process ensures that the limited cache size on the router is constantly updated with the most relevant and in-demand content. The dynamic interaction between content placement and replacement optimizes resource usage and enhances the user experience in the network.

In this chapter, we explore various caching techniques and classify them based on how they position content in the network routers. Caching methods in NDN can be categorized as centralized, decentralized, on-path, off-path, cooperative, non-cooperative, popularity-aware, probabilistic, etc. However, these methods often combine multiple approaches, making them difficult to classify into specific categories. For example, popularity-based methods may use both centralized and decentralized approaches, as well as on-path and off-path techniques.

In this chapter, we briefly describe the working philosophy of different caching techniques and then discuss existing work on NDN caching under two main categories: popularity-agnostic and popularity-aware.

## 2.1 Caching Techniques Taxonomy

Based on the nature of content retrieval and caching decisions, we classify existing ICN caching techniques into various categories. A comprehensive categorization of ICN caching strategies is illustrated in Figure 2.1. We first give an overview of each category and then elaborate on different works, taking reference to the category they belong to. Table 2.1

summarizes these works.

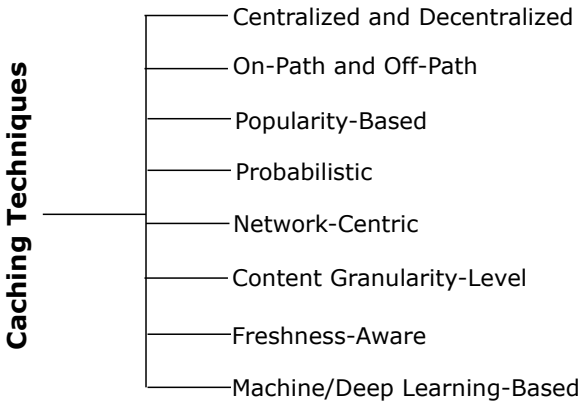



Figure 2.1: Taxonomy of NDN Caching.

*1)* ***Centralized and Decentralized Caching:*** This approach addresses how routers in the network engage in the decision-making process to cache or not to cache the content. It explores how routers actively manage caches, searching for content, deciding on content caching based on various factors, and making room for new content when the cache becomes full. The categorization encompasses various approaches, such as centralized, decentralized, Semi-decentralized, cooperative, and non-cooperative techniques that we discussed below.

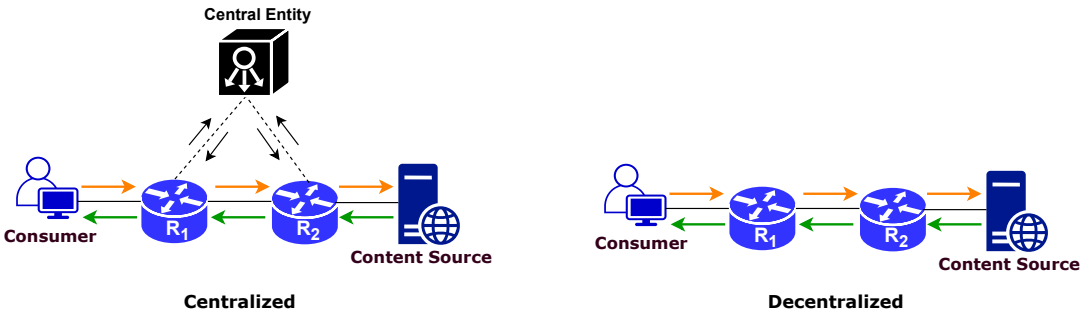



Figure 2.2: Centralized vs. Decentralized Caching.

**Centralized Caching:** Here, a central entity [110, 186] in the network assumes responsibility for making caching decisions for all other nodes or caching elements. For example,

in Figure 2.2, routers $R_1$ and $R_2$ are managed by the central entity. Centralized caching enables strategic content placement, facilitating network-wide resource optimization and efficient handling of off-path content requests. Although it enhances caching performance and resource utilization, challenges such as scalability, communication overhead, and potential single point of failure are a few issues with these methods.

**Decentralized Caching:** In a decentralized caching approach, the decision-making process is distributed across multiple routers in the network, allowing each router to make caching decisions [34, 123]. For example, in Figure 2.2, routers $R_1$ and $R_2$ make caching decisions without the help of any centralized entity. The decentralized approach enables routers to make content retrieval and caching decisions independently. Here, routers can either collaborate with peers to engage in collective decision-making or make caching decisions based solely on their local information. Decentralized decision-making among routers enhances adaptability to dynamic network conditions, thereby improving resilience and scalability. However, challenges may arise in maintaining coordination among autonomously operating routers to optimize cache utilization.

**Community-Centric Caching:** Here, the network topology is divided into multiple communities [168] to better utilize the storage capacity of the routers. This division can be based on relationships among nodes or nodes with common interests, ensuring that nodes within the same community share similar characteristics [63]. In Figure 2.3, the example topology is divided into two communities: one consisting of three routers $(R_1, R_2, R_3)$ and the other consisting of two routers $(R_4, R_5)$.

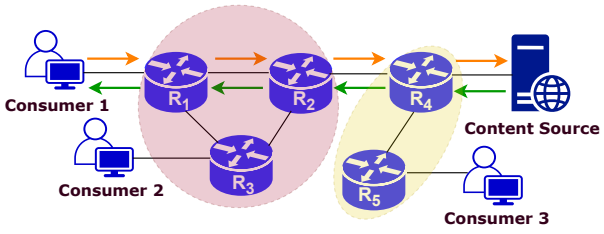



Figure 2.3: Community-Centric Caching.

In the community-centric approach, after forming the communities, certain node(s) are assigned the responsibility for content searching and caching. The designated node is elected based on its significance within the community to ease community management. This is a semi-decentralized approach, where nodes within each community are managed by a leader, while the communities operate independently in a distributed manner. The community-centric approach enhances cache utilization, reduces load imbalance in networks, and minimizes content redundancy through cooperation. However, challenges arise in forming communities and selecting representative nodes (in each community) to facilitate this cooperation.

**Cooperative Caching:** In a cooperative approach, routers within the network work together to optimize the limited cache storage [63, 90]. This collaborative effort improves content retrieval efficiency as routers share partial or complete cache state information with others to help them make decisions on caching. However, this comes with the additional cost of communication overhead. By strategically deciding on content caching, the cooperative approach contributes to the improvement of both the cache hit ratio and network-wide cache diversity. Moreover, the enhancement in content availability resulting from the collaborative approach enables routers to fulfill both on-path and off-path requests. The establishment of collaboration among routers can take place through either a centralized or decentralized decision-making approach. Depending on how routers cooperate, the collaborative methods can be classified into implicit and explicit techniques [128, 179].

*i) Implicit:* In implicit cooperation, routers do not send any explicit messages to exchange the cache state with others [179, 181]. Instead, cooperation can be achieved in a different way, such as by appending information to Interest or Data packets during routine communications. As shown in Figure 2.4, routers $R_1$ and $R_2$ coordinate indirectly through packet forwarding, without explicit signals or control messages. This minimizes communication costs as routers share limited information, improving content availability compared to non-cooperative techniques. However, this method fails to take full advantage of the content

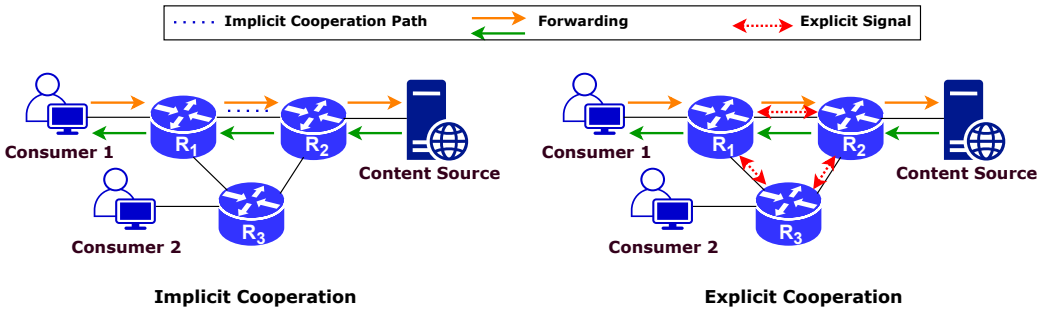



Figure 2.4: Implicit vs. Explicit Cooperation.

availability in the cache of neighboring routers.

*ii) Explicit:* In the explicit mode of cooperation, routers explicitly share cache state with other routers [47, 179]. The cache state can include either the availability of content, the popularity of content, or any other relevant information. In Figure 2.4, routers establish cooperation among themselves by sending explicit messages (indicated by the dotted red lines). Explicit cooperation proves beneficial in scenarios where a centralized controller [99] gathers information about routers in the network or when a cluster head establishes communication within or among multiple clusters. Additionally, it is advantageous for establishing communication within the immediate neighborhood (*i.e.,* 1-hop) and the multi-hop neighborhoods (*i.e.,* 2-hop, 3-hop, . . . , k-hop) [62], as well as for enabling cooperation among routers located along the transmission path. In explicit cooperation, routers have the flexibility to coordinate either at a local or global level with other routers in the network. Local collaboration involves coordination on the transmission path, within the neighborhood, or within specific domains. In contrast, global cooperation involves establishing coordination among routers to acquire knowledge of the entire network topology. Explicit collaboration enhances cache diversity and the availability of content for both on-path and off-path requests. However, it comes at the cost of increased communication overhead.

**Non-Cooperative Caching:** Here, each router makes independent decisions regarding the routing of Interest and Data packets [44, 182]. As shown in Figure 2.5, routers $R_1$ and $R_2$

forward packets between Consumer1 and the source without mutual coordination. Implementing a non-cooperative approach is simple and incurs lower communication overhead, as routers do not need to share their cache state with others. Despite these advantages, the non-cooperative approach encounters difficulties in attaining optimal resource utilization due to the limited cache size of the router. Routers, acting independently, may overlook the caching state of neighboring routers, resulting in content redundancy as multiple routers might cache the same content already stored by others. Moreover, the effectiveness of the non-cooperative approach declines as it overlooks broader request patterns.

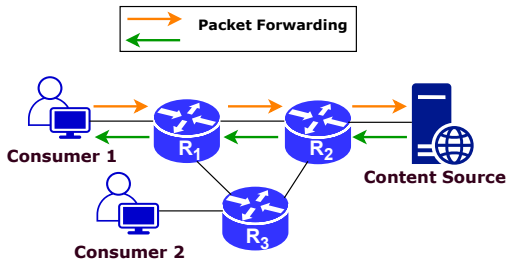



Figure 2.5: Non-Cooperative Caching.

*2)* ***On-Path and Off-Path Caching:*** Caching techniques are classified as on-path or off-path [45]. The classification depends on the location of the routers caching the content relative to the delivery path between the consumer and the provider. Intermediate routers positioned on the transmission path between the consumer and provider are termed as on-path routers, while any other routers not directly situated on the transmission path are considered off-path routers. This classification focuses on how routers strategically position content within their caches, influencing the efficiency of content retrieval and distribution across the network. For example, as shown in Figure 2.6, routers $R_1$ and $R_2$ are on-path routers between Consumer1 and the content source, whereas router $R_3$ is an off-path router because it is located outside the transmission path between Consumer1 and the content source.

**On-Path Caching:** On-path approach is inherent in the ICN/NDN architecture, where

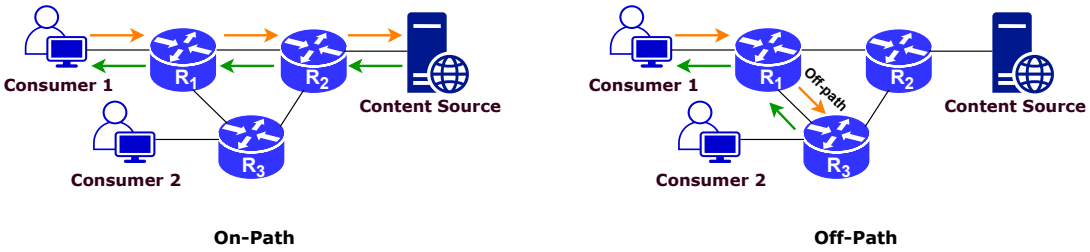



Figure 2.6: On-Path vs. Off-Path Caching.

Interests are forwarded via the best path to the content source, and Data packets take the reverse path created by the Interest [180]. Routers consult their FIB to guide the Interest packet to the next hop along the path towards the potential content provider, and only routers situated along this path query for the content [118]. Once the content is found, the Data packet follows the same path in the reverse direction, from the provider to the consumer. Only the routers along this route are eligible for content caching [7, 100]. In the downstream direction, content is either cached by all routers along the path or by a strategically selected subset of routers. For example, in Figure 2.6, Interest packets from Consumer1 are forwarded via routers $R_1$ and $R_2$ toward the content source. These two routers are on-path for this communication. In the reverse direction (downstream path), the content may be cached at $R_1$ and $R_2$ while being delivered to Consumer1. The implementation of on-path caching is simple and can be used with both cooperative and non-cooperative setups. On-path approach reduces both network and processing load, as only routers on the transmission path participate in the decision process. However, on-path techniques may face challenges with content availability, as they do not utilize content available off the path closer to the consumer.

**Off-Path Caching:** Off-path techniques [45] are more flexible than on-path techniques, as routers can query and cache content beyond the transmission path (*i.e.,* located at a distance of 1-hop, 2-hop, or even extending up to k-hop). Content can be served from any nearby router, either along or outside the request path [17, 85]. Similarly, content can be

cached at any location in the network [134]. Off-path caching techniques are beneficial when the on-path link becomes a bottleneck, as routers can retrieve content through an alternate path. As shown in Figure 2.6, with the off-path technique, if the content is available in the cache of $R_3$, the request is served from $R_3$ rather than being forwarded to the content source. This reduces the load on the source and decreases the number of hops required to retrieve the content.

Off-path caching can be accomplished through various methods. One approach incorporates a centralized SDN controller [110] with a global topology view that directs where content requests should be forwarded and cached. Alternatively, network topology partitioning into multiple clusters [170, 174] and electing cluster heads to enable off-path caching. Another approach includes the utilization of predefined rules (e.g., content retrieval and placement using the corresponding hash value) [134] or modifications to native NDN packet structures [62, 152]. NDN packets are modified by adding extra fields to Interest and Data packets to control requests and responses. For example, additional fields in the Interest packet help determine where to forward the request to retrieve content, while in the Data packet, they assist in making caching decisions. Off-path caching enhances content availability across the network and allows routers to retrieve content from any neighborhood routers. However, implementing off-path caching is complex and introduces communication overhead in order to leverage its benefits. In the literature, exploration of off-path caches is done through either reactive (on-demand) [32, 157] or proactive (table-driven) [86, 124, 156] schemes. In reactive schemes, requests are flooded [33] to discover off-path caches only when a user demands content, causing high overhead and congestion. In proactive schemes, routers exchange cache information in advance to know the content in their neighborhood, which may result in outdated entries, while frequent updates to keep content fresh lead to high signaling overhead and bandwidth consumption.

*3)* ***Popularity-Based Caching:*** Content popularity [19, 111] is measured by the frequency of consumer requests over time. The higher the access count, the more popular the content

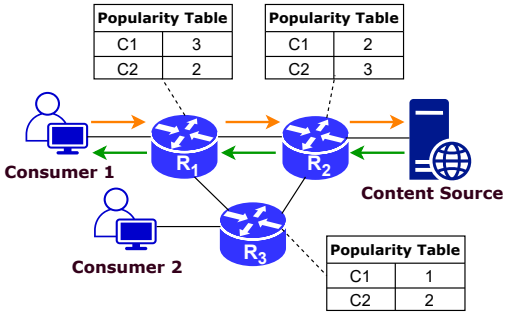



Figure 2.7: Popularity-Based Caching.

is considered. Unlike popularity-agnostic approaches, where routers make caching decisions without prioritizing any content, popularity-based schemes strategically cache content based on consumer demand. The content popularity metric assists routers in deciding which content to place in their caches and which ones to remove. By keeping frequently requested content closer to consumers, these schemes enhance performance and reduce the load on the original content source. As can be seen in Figure 2.7, each router maintains a table to record the history of requests made by consumers, which helps determine the popularity of the content. For example, in Figure 2.7, the popularity table at router $R_1$ shows that content C1 has been accessed three times, while C2 has been accessed twice. This means that at $R_1$, C1 is more popular than C2 because it has been accessed more frequently.

Popularity estimation by routers is classified as local or global according to the scope or region considered. Local estimation is based on requests observed within a single router or cluster/domain, whereas global estimation relies on aggregated requests from the entire network. The content is considered either locally or globally popular if its access count exceeds a certain threshold. This threshold can be either static or dynamic. Static thresholds are a pre-defined fixed value, while dynamic thresholds adapt based on observed request patterns. Figure 2.8 shows the taxonomy of popularity-based caching based on the popularity estimation scope and threshold policy.

**Local Popularity:** Local popularity estimation involves analyzing consumer content access

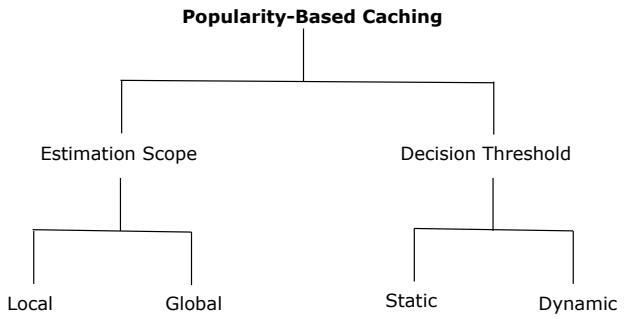



Figure 2.8: Taxonomy of Popularity-Based Caching.

patterns within specific regions. Each router keeps a popularity table [10, 111] to monitor requested content names and their access frequencies. For example, as shown in Figure 2.7, routers $R_1$, $R_2$, and $R_3$ maintain such a table to track the content passing through them. When a content request arrives, the router updates this table to make informed decisions on whether to cache the content in its CS. This localized approach enables routers to customize caching decisions to meet the specific needs of regional demands, thereby enhancing content availability and user experience while minimizing coordination overhead. However, the challenge with this caching approach arises from its inclination toward local content preferences, which neglects broader demand from diverse consumer regions and may reduce caching performance.

**Global Popularity:** This method assesses the overall popularity of content across the entire network, improving network-wide content availability through strategic caching of globally popular content [50, 165]. Routers store frequently requested content in their cache by considering the combined demand from diverse consumer regions instead of specific ones. Global popularity estimation can be done either through a central entity [186] or a collaborative approach among routers [50]. In the centralized approach, a central entity (e.g., an SDN controller) collects local popularity information from routers to obtain a global view of popularity. In the collaborative approach, routers exchange their local popularity information with each other or with designated routers to obtain a view of global popularity. This process

can be executed at specified intervals, after a router sees a certain number of requests, or on demand when required. While improving network-wide content availability and response time, estimating global popularity poses challenges, such as coordinating among nodes to track request counts and manage the popularity table. This may lead to increased communication and computational overhead, especially in large-scale networks with diverse user behaviors.

As shown in Figure 2.8, based on the scope or region considered by the router, content popularity is estimated as either local or global. These popularity estimation methods use either static or dynamic thresholds to determine whether content is popular.

**Static Popularity:** In this approach, content popularity is determined by analyzing historical request patterns or adhering to predefined thresholds [5, 19, 101]. The assumption here is that the content's popularity remains consistent over a specific time period or adheres to a predictable trend. For example, caching a highly demanded video during peak periods at important network locations. In the static method, content with request frequency above the set threshold is marked as popular, while content below it is considered unpopular and may not be cached. This technique is simple and requires less effort as routers don't have to frequently modify the threshold values, making it suitable for stable content popularity trends. However, network performance diminishes as content popularity dynamically fluctuates over time in real-world scenarios, and it presents challenges in adapting to the diverse requests of consumers over time.

**Dynamic Popularity:** Dynamic popularity estimation captures the real-time changes in content popularity based on current demand [7, 50]. This approach involves routers considering various real-time factors, such as the total request count during a specific time interval, cache size, content size, etc., to calculate a dynamic popularity threshold value[111]. This value determines the popularity status of content at a specific router or within a particular region. Unlike a static threshold, the dynamic threshold may vary between routers or differ across various regions within a network. The dynamic threshold value fluctuates over time

in response to changing content trends, offering a more responsive and accurate estimation of content popularity. For example, a specific news article gains popularity during a certain period and subsequently declines in popularity. However, dynamic popularity estimation increases both communication and computational overhead, as routers need to periodically adjust the threshold value to track real-time changes.

*4)* ***Probabilistic Caching:*** These methods use a probability [93, 118] value to make content caching decisions instead of relying on deterministic rules. Routers can either use predetermined fixed probability values or random values within the interval of 0 to 1 for making caching decisions. For example, if the probability value is set to 0.5, the router caches the content only when the probability at that router reaches the threshold. Alternatively, routers can use a dynamic probabilistic approach, considering dynamic factors such as network traffic, cache capacity, and the distance between the consumer and provider to make content caching decisions.

*5)* ***Network-Centric Caching:*** In this approach, content is stored within the network based on the topological structure and characteristics of routers, such as their number, location, connectivity, link latency, etc. The goal is to cache content close to the consumer in order to reduce access time. Strategically placing content at important locations in the network enhances cache hits and reduces the load on the original content source. Based on graph-based metrics, these caching schemes evaluate the importance of routers within the network topology [8, 74]. Key metrics, such as degree centrality, closeness centrality, and betweenness centrality [108, 115, 190], play a crucial role in shaping caching decisions. These metrics help in placing and retrieving content strategically to improve network performance.

At the network level, these caching schemes use either a consistent strategy for caching content at both the edge (closer to the consumer) and core (closer to the content source) of the network or implement a different approach based on specific requirements [7]. In edge caching, content is stored at the network edge or in routers directly connected to consumers. On the other hand, core caching involves storing content in routers positioned at the core of

the network, ensuring availability to a broader range of consumers across different regions.

In order to enhance the efficiency of searching and caching operations, the network topology is viewed in smaller regions [175] rather than the entire topology. This approach simplifies topology management and improves the understanding of consumer demand trends. One approach to implementing this involves structuring the network topology into multiple clusters or domains [63, 175]. Another viable method is a neighborhood-based approach, establishing communication among neighbors within 1-hop, 2-hop, or k-hop distances [62].

*6)* ***Content Granularity Level Caching:*** The level of information or amount of content kept in a cache is referred to as content granularity. This concept entails the identification of the specific content or information to be stored in the cache, aiming to enhance the efficiency of content retrieval. The granularity can vary, involving fine-grained approaches like chunk-level caching [34, 126], where individual content chunks or blocks are cached, and coarse-grained strategies like object-level caching, where entire requested content objects or files are considered for caching [114]. Another level of granularity, referred to as packet-level caching [12, 151], manages objects with finer granularity than object-level, which involves caching at the level of individual network packets. The decision regarding the extent of information to be stored in the cache is influenced by the requirements of the caching methodology.

*7)* ***Freshness-Aware Caching:*** Content freshness is one of the most important factors in caching. These strategies consider the age of content. The content lifetime is defined with respect to the time when it is created by the content provider until its expiration time [188]. The content freshness parameter [120, 185] in the Data packet is set to indicate the lifetime of the content. During this time, routers can hold the content in their cache. Content freshness [4, 7] indicates how recent the content is at the time of caching.

*8)* ***Machine/Deep Learning-Based Caching:*** In recent years, ML/DL algorithms have been widely used in various areas of networking. For example, these algorithms are used to predict advanced cyber threats, detect unusual network behaviors, monitor networks in real time, optimize resource allocation, and many other applications. Along similar lines,

researchers [54, 56, 97, 189] have used ML/DL techniques to make adaptive caching decisions in NDN to improve its performance. These adaptive algorithms can help make caching decisions based on real-time changes in user demand and network conditions. These algorithms help predict content demand and identify the best locations for caching to maximize benefits. Popular algorithms, such as Graph Neural Networks (GNN) [56, 57] and Reinforcement Learning (RL) [66, 189], are used to predict user demand and network conditions to select the most appropriate content and locations for caching. However, due to the complexity of these algorithms, conventional content caching methods remain popular choices. This is because the router can make caching decisions quickly while forwarding Data packets towards consumers, improving content availability and reducing response time.

Table 2.1: Summary of Various Caching Categories

| **Caching Category** | | | **Working Principle** | **Works** |
|---|---|---|---|---|
| Centralized | | | The controller(s) or designated node(s) centrally manage content lookup and caching decisions. | [6], [72], [186] |
| Decentralized | | | Distributed decision-making empowers multiple routers to perform content searching and caching decisions autonomously. | [34], [123] |
| On-path | | | Content searching and caching are confined to routers situated on the communication path between the consumer and provider. | [45], [70], [84], [118] |
| Off-path | | | Provides flexibility in content exploration and caching. In the upstream direction, content can be retrieved from any router, and in the downstream direction, content may be cached at any router in the network. | [35], [45], [134], [171] |
| Non-cooperative | | | Routers independently perform content lookup and caching operations without collaboration. | [70], [182] |
| Cooperative | Implicit | | Routers do not directly share content availability or any other information between them. | [8], [34], [83], [89] |
| | Explicit | | Routers explicitly advertise their content availability and share other information. | [62], [156] |
| Popularity-Agnostic | | | Treating all content equally, irrespective of the frequency of requests. | [29], [47], [53] |
| Popularity-Based | Scope/ Region | Local | Considers consumer requests in a specific region, or each router individually tracks requests passing through it. | [19], [111] |

| Caching Category (cont.) | | | Working Principle | Works |
|---|---|---|---|---|
| | | Global | Considers content requests from all consumers across the entire network. | [50], [90] |
| | Threshold | Static | Utilizes a fixed, predefined threshold value to determine which content to cache. | [19], [111], [101] |
| | | Dynamic | Adjusts the threshold value dynamically based on changing network conditions. | [111] |
| Probabilistic-Based | Fixed | | Routers make caching decisions using a predetermined fixed probability value $p$. | [83], [150], [181] |
| | Dynamic | | Routers dynamically calculate the value of $p$ based on factors such as popularity, cache size, network traffic, distance, etc. | [52], [118], [149], [161] |
| Network-Centric | Edge Caching | | Storing content at the network edge or the first upstream router to the consumer. | [5], [135] |
| | Single Node | | During content delivery, caching occurs at a single router in the network. | [83], [134] |
| | Multi-Node | | During content delivery, caching occurs at multiple routers in the network. | [70], [84] |
| | Topology-Aware | | Optimizes caching and retrieval by considering network topology attributes, including connectivity, link latency, and centrality metrics like degree, closeness, and betweenness. Routers with higher centrality values are prioritized for caching. | [8], [27], [75], [130], [190] |
| | Location-Aware | | Cache and retrieve content from specified routers in the network or domains. | [134], [145] |
| | Neighborhood-Aware | | Caching and retrieval operations are influenced by the proximity of routers. | [62], [103], [112] |
| | Community-Aware | | Viewing network topology as multiple community or cluster structures based on node relationships for optimized content searching and caching decisions. | [63], [145], [168], [170] |
| Content Granularity | Object-Level | | Routers cache entire objects or files. | [114] |
| | Packet-Level | | Caching data at the level of individual network packets. | [12], [70], [151] |
| | Chunk-Level | | Routers cache large files by dividing them into chunks or blocks. | [34], [92], [95] |
| | Network Coded | | Chunks are encoded for caching in the router to improve data transmission efficiency. | [62], [105], [136], [167] |
| Freshness-Aware | | | Content remains valid until the freshness period expires. | [4], [5], [7], [120] |

| Caching Category (cont.) | Working Principle | Works |
|---|---|---|
| Machine & Deep Learning-Based | Using advanced learning methods like deep learning, reinforcement learning, and neural networks, routers can predict consumer needs, optimize content caching and retrieval, and adapt to real-time changes in user behavior. | [55], [57], [66], [71], [97] |

## 2.2 Overview of Existing Caching Strategies

In this section, we describe several important caching techniques designed by the ICN/NDN community. Several caching methods do not fall exclusively into a single type, as mentioned previously. Therefore, we chose to elaborate on specific works, highlighting the features each method possesses. We categorize caching methods into two broad categories: popularity-agnostic methods, which do not consider content popularity, and popularity-aware methods, which take content popularity into account. Caching techniques within both the popularity-agnostic and popularity-aware categories can employ one or more combinations of the previously discussed approaches, such as centralized or decentralized, on-path or off-path, and cooperative or non-cooperative. We elaborate on the working of different approaches of the two categories by taking reference to a small topology shown in Figure 2.9. Table 2.2 presents the summary of existing caching techniques under different categories. A tick mark (✓) in the table indicates that the caching technique falls under that category.

### 2.2.1 Popularity-Agnostic Caching

These caching methods do not consider the frequency of requests while making caching decisions.

**Leave Copy Everywhere (LCE) [70]**: LCE is alternatively referred to as Cache Everything Everywhere (CEE). This is a non-cooperative on-path caching approach, where routers positioned between the content consumer and provider (router/content original source) cache all content passing through them in the downstream direction. LCE involves caching the same content in all routers along the transmission path. For

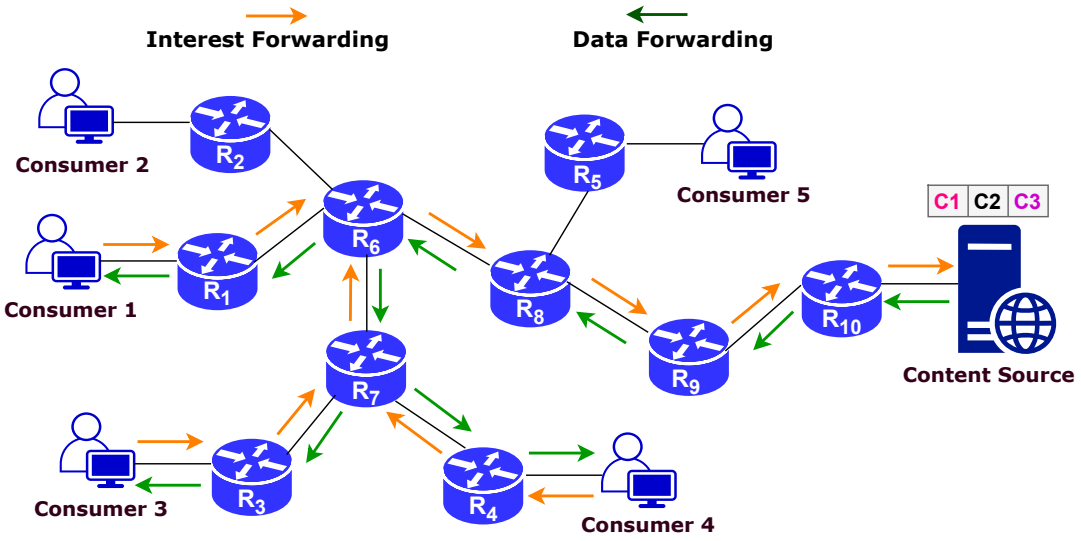



Figure 2.9: Example Topology

example, in Figure 2.9, Consumer1 initiates a request (C1), which is served by the content source. The request is routed via the shortest path through routers $R_1$, $R_6$, $R_8$, $R_9$, and $R_{10}$ to reach the provider. In response, content C1 is cached at $R_{10}$, $R_9$, $R_8$, $R_6$, and $R_1$, then served to Consumer1. Implementing LCE is straightforward, as routers do not interact with neighbors in caching decisions. However, caching the same content at multiple routers increases redundancy and eviction rates, impacting the overall performance.

**Leave Copy Edge (LCEdge) [135]**: LCEdge is a single-node on-path caching approach that caches content at the last router in the downstream direction, typically one hop away from the consumer, especially when the directly connected router to the consumer is cache-enabled. In Figure 2.9, if Consumer1 requests content C1, it is provided by the source node. In the downstream direction, the content is cached only at router $R_1$, while other routers on the path forward the packet to the next hop without caching. LCEdge is designed to minimize content access time by keeping content closer to the consumer. However, it underutilizes other routers on the delivery path that are also capable of holding content.

**Leave Copy Down (LCD) [83]**: LCD is another popular on-path caching technique

Table 2.2: Caching Technique Classification

| Caching Technique | Centralized | Decentralized | On-path | Off-path | Cooperative | | Non-Cooperative | Popularity |
|---|---|---|---|---|---|---|---|---|
| | | | | | Implicit | Explicit | | |
| LCE or CEE [70] | × | ✓ | ✓ | × | × | × | ✓ | × |
| LCD [83] | × | ✓ | ✓ | × | ✓ | × | × | × |
| MCD [83] | × | ✓ | ✓ | × | ✓ | × | × | × |
| Prob(p) [150] | × | ✓ | ✓ | × | × | × | ✓ | × |
| Prob-PD [64] | × | ✓ | ✓ | × | ✓ | × | × | ✓ |
| ProbCache [118] | × | ✓ | ✓ | × | ✓ | × | × | × |
| MPC [19] | × | ✓ | ✓ | ✓ | × | ✓ | × | ✓ |
| FGPC/D-FGPC [111] | × | ✓ | ✓ | × | × | × | ✓ | ✓ |
| Intra-AS [156] | × | ✓ | ✓ | ✓ | × | ✓ | × | × |
| O2CEMF [62] | × | ✓ | ✓ | ✓ | × | ✓ | × | × |
| NACID [112] | × | ✓ | ✓ | ✓ | × | ✓ | × | ✓ |
| Hash-Routing (HR) [134] | × | ✓ | ✓ | ✓ | × | × | ✓ | × |
| CPHR [158] | × | ✓ | ✓ | ✓ | ✓ | × | × | ✓ |
| BEP [190] | × | ✓ | ✓ | × | ✓ | × | × | ✓ |
| OpenCache [171] | × | ✓ | ✓ | ✓ | × | ✓ | × | ✓ |
| OFAM-CCN [72] | ✓ | × | ✓ | ✓ | × | ✓ | × | ✓ |
| Compound Popularity [50] | × | ✓ | ✓ | × | × | ✓ | × | ✓ |
| SDN-Based Caching [186] | ✓ | × | ✓ | ✓ | × | ✓ | × | ✓ |
| CCS/CES [7] | × | ✓ | ✓ | × | ✓ | × | × | ✓ |
| SDCC [140] | ✓ | ✓ | ✓ | ✓ | × | ✓ | × | ✓ |
| PF-EdgeCache [5] | × | ✓ | ✓ | × | × | × | ✓ | ✓ |
| PTF [166] | × | ✓ | ✓ | × | ✓ | × | × | ✓ |
| SoftCaching [121] | ✓ | × | ✓ | ✓ | × | ✓ | × | × |
| EABC [53] | × | ✓ | ✓ | × | ✓ | × | × | × |
| CRUS [93] | × | ✓ | ✓ | × | ✓ | × | × | ✓ |

specifically designed to address the content redundancy challenges posed by LCE. Unlike LCE, LCD strategically caches content at only one router along the delivery path. In the case of LCD, the content is cached one hop down from the content provider in the downstream direction, while other routers along the path forward the packet without caching it. The effectiveness of LCD relies on implicit cooperation with the next hop router, facilitated by the inclusion of a flag bit in the packet. This flag informs routers on the forwarding path that the upper-level router has already cached the content. LCD brings frequently requested content closer to the consumer upon subsequent hits. It achieves this without the need to monitor content popularity through the maintenance of additional tables. Instead, it intuitively brings popular content closer to consumers with each subsequent hit, effectively increasing the hit

ratio and minimizing content access time. To illustrate this concept, let's refer to Figure 2.9. In the initial round, Consumer1 requests content C1, which is initially available at the source node. As it travels downstream, C1 is cached one hop down at $R_{10}$. In the second round, Consumer1 again requests C1, which is now provided by $R_{10}$ and cached at $R_9$, and this is repeated until the content is brought to $R_1$ through successive requests. However, it becomes apparent that the same content is cached at all routers along the transmission path, leading to content redundancy similar to the LCE. Another concern with LCD is that if consumers request different content each time, for example, in a round-robin fashion, caching and eviction operations only occur at a specific router (e.g., $R_{10}$). This can result in a reduced cache hit ratio and increased content access time.

**Move Copy Down (MCD) [83]**: MCD is another strategy that caches content at a single router along the delivery path. MCD reduces redundancy compared to LCD by deleting content from the provider node after caching it one hop down the router in the downstream direction, except when the content is served by the original source. However, a challenge with MCD is that it may move the content from important routers, *i.e.,* routers that lie on multiple shortest paths between consumers and providers. For example, consider the network topology shown in Figure 2.9. Let us assume that content C1 is available in the cache of $R_6$. When Consumer1 requests C1, it is served directly by $R_6$. According to the MCD caching policy, C1 is moved from $R_6$ to $R_1$, because MCD moves content one hop down from the cache-hit router. $R_6$ is an important router in the topology, as it lies on the path for many consumers. Moving cached content from this router impacts performance. For example, if Consumer2 now requests C1, the request must traverse $R_2 \rightarrow R_6 \rightarrow R_8 \rightarrow R_9 \rightarrow R_{10}$ to reach the content source. This increases the number of hops required to retrieve C1.

**Probabilistic-Caching:** As mentioned in Section 2.1, probabilistic caching methods derive a probability value to make caching decisions. The studies in [118, 150] adopt probabilistic caching, where a router $R$ decides whether to cache content C based on a probability $p \in [0, 1]$.

*Prob(p) [83, 150]:* Prob(p) enhances the conventional approach (caching everything

along all routers in the downstream direction) by selectively caching content with a fixed probability value $p$. This probability-based caching significantly decreases content redundancy. In the downstream path, each router independently generates a random number between 0 and 1. If this generated number is less than $p$, the router caches the content; otherwise, it refrains from caching. Like LCE, Prob(p) operates in a non-cooperative manner, with each on-path router independently making caching decisions. This decentralized decision-making process avoids communication overhead. Notably, Prob(p) significantly reduces content redundancy and replacement rates compared to LCE. The effectiveness of Prob(p) depends on the predefined value of $p$; when $p$ reaches 1, Prob(p) becomes equivalent to LCE. Because of this adaptability, Prob(p) can exhibit varied performance characteristics depending on the given probability threshold.

*ProbCache [118]:* ProbCache is another popular on-path probabilistic caching method that caches content at a router selected based on its position or importance along the content delivery path. ProbCache brings content closer to the consumer based on the estimated probability. In order to monitor the number of hops the request and content packets travel, the ProbCache introduces two additional fields in the ICN packet structure: Time Since Inception (TSI) and Time Since Birth (TSB). The TSI field in the request message indicates the hop-distance from the client (*i.e.,* how far the request has traveled from the user), whereas the TSB field in the content message header indicates the hop-distance from the server (*i.e.,* how far the content has traveled from the source). In the upstream direction, the TSI value increases by one at each crossing router, while in the return path, the TSI value remains fixed, reflecting the distance between the consumer and provider. Simultaneously, in the downstream direction, the TSB value increments at each crossing router. By utilizing TSI and TSB values, the ProbCache calculates `TimesIn`, representing the number of times the path can accommodate caching for a given packet. It also considers the cache weight factor to ensure a fair distribution of resources. Each router makes caching decisions based on the estimated probability, determined by the product of `TimesIn` and the cache weight factors. To enhance the performance of the ProbCache

[118] strategy, an extension called ProbCache+ [119] was introduced. ProbCache+ improves performance by modifying the cache weight factor.

ProbCache uses implicit cooperation techniques wherein routers do not externally advertise their cache status to neighboring routers. This approach reduces communication overhead. Caching content near the consumer along the on-path route increases the hit rate. However, a limitation of ProbCache is its lack of consideration for the popularity level of content in the network, making it unable to distinguish between popular and unpopular content. This absence of content popularity distinction may potentially impact its overall performance.

**Neighborhood-Based Cooperative Caching:** As discussed previously, cooperative caching establishes coordination in the neighborhood for content searching and caching. The works presented in [62, 156] strategically explore nearby off-path routers to retrieve available content while forwarding requests towards the provider.

Hu *et al.* [62] proposed an on-demand off-path cache exploration technique termed O2CEMF for the network coding enabled NDN. Network coding divides large files into multiple chunks, which are then encoded before being exchanged between network nodes. This enables multi-path forwarding of several chunks simultaneously. O2CEMF discovers encoded blocks present in off-path routers while forwarding request packets to the original content source. It uses a multipath packet transmission approach in which Interest packets are simultaneously forwarded to both upstream on-path and off-path routers. O2CEMF operates in two phases: exploration and exploitation. During the exploration phase, each router situated on the request forwarding path sends O2CEMF Interest packets to all its neighboring routers. The off-path routers respond with either encoded block information from their caches or a NACK packet if there are insufficient blocks available for the requested generation. Using this information, the router constructs the cache trails table, which proves beneficial in the exploitation phase. In the exploitation phase, routers send Interests based on matching entries in the cache trails. However, due to the high volatility of cached content, O2CEMF may introduce signaling overhead due to the flooding of Interest packets.

Wang *et al.* [156] proposed an off-path content searching method that explores up

to a 1-hop neighborhood to retrieve requested content before forwarding the request to the original source. However, this is limited to within the Intra-Autonomous System (Intra-AS), enabling routers to probe within this Intra-AS framework. Neighboring routers collaborate to satisfy each other's requests while also reducing content redundancy in the neighborhood. Routers periodically share their cache table information with directly connected routers using the Bloom filter data structure, effectively minimizing the overhead associated with information sharing. Another caching technique [106] uses the Bloom filter data structure to store cache information of 1-hop neighbors by sharing cache summaries among routers. Each router performs Bloom filter lookups while forwarding both Interest and Data packets, which helps retrieve content from nearby caches and reduces content redundancy in the neighborhood. When the Data packet is received, the router checks the Bloom filter, which contains a summary of the content cached by its neighbors. If the lookup is positive, the content is forwarded without caching; otherwise, the content is cached.

Badshah *et al.* [14] proposed a centralized caching method for SDN-based ICN architecture. In this solution, multiple cache servers are deployed in the topology instead of a single centralized cache server [76] to distribute the load and enhance system efficiency. The proposed approach begins by calculating the required number of cache servers based on the size of the network topology. Subsequently, it deploys these cache servers strategically, considering multiple metrics such as path stretch, closeness centrality, and betweenness centrality. This strategic placement aims to reduce traffic and content access time by situating cache servers closer to routers. Once the cache servers are appropriately located, the network controller is responsible for installing flow rules in every switch to direct requests to the nearest cache server. The controller, acting as a global entity, keeps track of cache server and content provider locations in the network. Upon receiving a request, if the requested content is found in the cache server, it is returned to the requesting node. However, if the content is not found in the cache, the request is then forwarded to the controller. The controller then determines the path between the cache server and the appropriate content provider for handling the request and content routing. In the response path, the content provider first sends

the content to the cache server for caching purposes. Subsequently, the cache server forwards the content to the requesting node.

Rafique *et al.* [121] proposed a centralized approach known as SoftCaching for IoT networks, which integrates SDN with ICN to efficiently perform the routing and caching operations. With the help of a controller, content is cached at selected routers, which are considered the best for caching. SoftCaching focuses on selecting the best caching nodes based on network traffic analysis. The controller, possessing a global view of the network, plays a crucial role in identifying caching nodes, installing flow rules in switches for routing requests to the nearest caching node, and overseeing all ICN-related operations. In the SoftCaching framework, the choice of best nodes for caching is determined through an analysis of network traffic, using methods such as Singular Value Decomposition (SVD) and QR Factorization. SoftCaching aims to improve the effectiveness of caching by strategically choosing caching nodes through a thorough assessment of network traffic patterns.

Authors of [85] designed a caching technique called Heartbeat for NDN to access content available in the caches of off-path routers. Heartbeat uses a proactive advertisement approach that enables effective utilization of off-path caches. However, it employs a lightweight mechanism to share content availability information among neighboring nodes. Instead of sending entire catalogs periodically, Heartbeat sends virtual interests to neighboring nodes whenever content is inserted or relocated in a cache. Neighboring nodes use this virtual interest to create temporary FIB entries with carefully chosen lifetimes. This enables user requests to be redirected to nearby off-path caches without causing flooding. The simulation results demonstrate that Heartbeat improves cache utilization while reducing bandwidth consumption and overhead.

**Hash-Based Techniques:** Hash-based methods utilize a hash function to map requested content identifiers to hash values, enabling the identification of routers within the network. This location-aware method allows routers to determine where to forward Interest packets and where to cache the content in the network. Different hash-based techniques [91, 132, 134, 138, 145, 158] have been designed to locate content in the

network. These techniques fall under the category of off-path caching approaches, as content can be retrieved and cached from anywhere in the network, not necessarily always on the transmission path.

Saino *et al.* [134] introduced a hash-based technique that assigns a unique ID in the range of 1 to N to every router within the domain or topology, where N denotes the total number of routers. For every unique content, one router is designated to cache the content in its CS based on the hash value of the request. When an edge router receives a request for content, it first calculates the hash value of the requested content by passing the request identifier to a hash function. This hash value is used for locating the designated cache node. Subsequently, the edge router routes the request packet to the designated cache node via the shortest path. If the content is available at the designated cache node, it sends a reply with the content; otherwise, the request packet is forwarded all the way to the source. Saino *et al.* evaluated five hash-based schemes: Symmetric, Asymmetric, Multicast, Hybrid Symmetric-Multicast, and Asymmetric-Multicast. While these techniques use the same routing mechanism for forwarding content requests, they differ in the way they forward content packets. The Symmetric technique follows the same path for both request and content forwarding. In this approach, the content is first forwarded to the designated node for caching and then sent to the requested edge router. In the Asymmetric scheme, the content packet follows the shortest path between the provider and the edge router. If the designated caching node is located on the shortest path, it caches the content; otherwise, this technique ignores caching. The Multicast approach involves forwarding one copy of the content to the designated node for caching and another copy to the requested edge router via the shortest path. Hybrid techniques combine the best aspects of these methods to enhance both routing and caching performance.

Hash-based schemes make use of the content available at off-path routers to improve the cache hit ratio and reduce cache redundancy by caching a single copy of the content in a network [134]. However, these schemes suffer from increased content access time in larger network domains/topologies due to the designated node potentially being distant from the edge router. To overcome these challenges, Sourlas *et al.* [145]

proposed a hash-based routing approach that incorporates domain clustering methods. This approach reduces content access time caused by hash-based routing schemes [134] while maintaining a higher cache hit ratio. In this technique, larger domains are divided into several clusters or groups, and a modulo hash function is used to map content requests to the designated cache node within the respective cluster. They use well-known clustering algorithms such as k-split and k-medoids to partition the large network topologies.

Wang *et al.* [158] presented a cooperative hash-based routing and caching scheme named CPHR with the objective of maximizing the cache hit ratio by eliminating content redundancy within the network. CPHR partitions the content space based on hash values and assigns each partition to the cache-enabled routers in the network. The process of forwarding Interest and content packets in CPHR follows a symmetric scheme, similar to the approach outlined in [134]. To enhance request forwarding, CPHR augments the FIB table by introducing an extra field called `EgressRouter`. This field guides the routing of Interests associated with a specific name prefix, directing them to the designated egress router before exiting the domain when forwarded towards the source node. Furthermore, each ingress router maintains a partition table to identify the partition of the received request and the corresponding assigned cache router. CPHR also adds two additional fields to the NDN Interest packet: `CacheName`, indicating the name of the assigned cache, and `EgressName`, denoting the name of the egress router. The assigned cache router updates the `EgressName` field set by the ingress router (edge router) to guide the Interest towards the original content source.

**Centrality-Based Caching:** As discussed in Section 2.1, centrality-based or topology-aware caching places content on selected important routers in the network to improve response time for end users.

Chai *et al.* [27] designed a caching technique for CCN that stores content only in selected important routers, rather than caching content at every router along the content delivery path. They used the betweenness centrality metric to identify strategically important routers along the delivery path, which are responsible for caching the content. This centrality-based scheme caches less content but provides greater

benefits in terms of cache hit ratio and lower latency compared to random caching or caching content at all routers along the delivery path.

Energy-aware Approximate Betweenness Centrality (EABC) [53] is a caching technique designed for NDN-based IoT networks that considers router centrality and energy availability when making caching decisions. In request forwarding, the EABC strategy adds a `Betw` field to the Interest packet, which helps to select the router with a higher betweenness centrality along the path. When an Interest packet passes through a router, each router along the path compares its centrality value with the value present in the `Betw` field. If the router's centrality value is higher than the `Betw` value, it updates the `Betw` field; otherwise, it forwards the Interest packet to the next hop without updating. In content forwarding, the EABC strategy adds two fields, `Betw` and `EnergyFlag`, to the Data packet to make caching decisions based on the router's centrality and energy level. The `Betw` field carries the centrality value of the router, and the `EnergyFlag` is used to determine whether the packet is new (sufficient energy) or old (insufficient energy). `EnergyFlag` value is set to 0 at the beginning, indicating that the packet is new. On subsequent hops, the value changes depending on the energy level of the router. The value of the `EnergyFlag` is set to 0 if the router's energy is higher than a predefined threshold; otherwise, it is set to 1, indicating that the router will not cache the content due to insufficient energy. At each hop, if the router's energy is lower than the predefined threshold, the value of the `EnergyFlag` is incremented by 1. When the `EnergyFlag` value exceeds 2, the content will be cached regardless of any conditions to improve the performance of caching.

### 2.2.2 Popularity-Based Caching

These caching schemes incorporate content popularity (access frequency) as one of the key factors while making caching decisions.

**Most Popular Content (MPC) [19]:** MPC proposes an approach to cache only popular content to enhance the cache hit ratio. Each router maintains a Popularity Table (PT) recording content names and their access counts. If an entry already exists, the counter is incremented. MPC relies on a predefined threshold to determine

local content popularity. If the request count surpasses this threshold, the content is considered popular and it is cached. The router then signals its neighbors to cache this popular content. The caching decision in the neighbor depends on the local policy (whether to cache or not). After making suggestions to neighbors, the router resets the popularity count to prevent unnecessary flooding of suggestions for the same content. In Figure 2.9, Consumer1 requests C1, which is available at the source node. For instance, at router $R_6$, the count of content C1 is 5, surpassing the set threshold of 5, categorizing C1 as popular and prompting its caching at $R_6$. Subsequently, $R_6$ suggests its neighbors, $R_1$, $R_2$, and $R_7$, to cache C1. This approach enables MPC to cache content at routers beyond the delivery path, like $R_2$ and $R_7$. Although MPC minimizes overhead by limiting popularity measures locally, caching only popular content may not optimally utilize the available cache space. Additionally, MPC might introduce overhead when disseminating caching suggestions to neighboring routers.

**Fine-Grained Popularity-Based Caching (FGPC) [111]:** Depending on the available cache space, FGPC strategically caches both unpopular and popular content to optimize cache utilization and hit rate. FGPC caches all passing content when space is available, and when the cache is full, it removes unpopular content while caching popular content using the Least Recently Used (LRU) replacement policy. To monitor the content frequency, each router maintains a Popularity Table (PT), similar to the MPC [19]. Content is considered locally popular to cache when the local count value of requested content reaches a predefined threshold. As content popularity changes over time, a static threshold may diminish performance. To address this, FGPC has been extended to D-FGPC [111], which caches content based on a dynamic threshold. This dynamic threshold adjusts depending on the available cache space, content size, and the request count of content in the PT at time T. However, the dynamic threshold introduces increased overhead at the routers.

**Probabilistic-Caching:** Ioannou *et al.* [64] introduced the on-path probabilistic caching scheme termed Prob-PD to enhance the performance of on-path caching. While the content is delivered in the reverse path, on-path routers dynamically calculate probabilities based on two key factors: the popularity of the requested content at

the router, making the caching decision, and the distance ratio of the router from the content source. Another work [159] models the caching decision using a probabilistic approach. The proposed scheme, CRCache, caches content in a few selected routers along the delivery path by considering the correlation between network topology and content popularity. Each router caches content with a probability $P$, where $P = 1-C$, and $C$ represents the correlation value. Every Data packet carries a *Content-Attached Popularity* (CAP) value. When the cache at router $R_i$ becomes full, a new content item is cached only if its CAP is higher than the lowest CAP among all contents stored at $R_i$ by replacing the lowest CAP content. Li *et al.* [93] proposed a probabilistic caching technique called the Caching-Resource Utilization-Based Strategy (CRUS). In this approach, the on-path router with the lowest resource utilization is selected for content placement. The cache resource utilization in CRUS is estimated based on the number of requests served by a router, $R_i$, relative to its cache occupancy. Content is cached in the on-path router based on probability. If the content is served by the source, the probability of caching is 1. However, if the content is served from the cache of a router, the caching probability is determined by comparing the number of hops between the selected router for content placement and the provider node to the number of hops between the provider and the consumer. The closer the selected router is to the consumer, the higher the probability of caching the content.

**Neighborhood-Based Cooperative Caching:** Work [171] introduced OpenCache, a lightweight collaborative caching method to retrieve popular content from nearby locations. The main idea behind this method is the exchange of hierarchical name prefixes among routers. By sharing these prefixes, routers make popular content accessible to other routers within the network. Neighboring routers then maintain a record of these shared prefixes along with their corresponding interfaces, directing subsequent requests to the nearby available caches. OpenCache achieves lightweight off-path cache collaboration through the extension of the original FIB with open cache entries. These entries encompass both open data prefixes and associated cache interfaces.

Another work [124] proposed a neighborhood collaboration scheme to achieve ef-

ficient content distribution by forming local clusters among adjacent routers (1-hop neighbors). Each router shares its cache information with neighboring routers whenever new content is cached or replaced by sending a *Content-Ad* message. Each router maintains a Neighbor Cache Table (NCT) to keep track of content availability in adjacent routers and the cost of fetching that content. The NCT helps construct the FIB table to retrieve content from nearby routers located outside the transmission path. While forwarding a request towards the provider, the router first checks its cache; if the content is not available, it checks for the content in any adjacent router. If the content is not available in any neighboring routers, the request is routed towards the provider. In the downstream direction, each router decides whether to cache content based on the state (True/False) of the *NeighborCached* flag, which is added to the Data packet and a threshold value. This flag helps avoid caching the same content in adjacent routers. The caching threshold is computed by considering several key factors, such as router connectivity, content popularity, hop count, and cache replacement rate, which help in caching the popular content at important routers. This requires additional storage and may not be feasible for routers with many neighbors. Additionally, periodically exchanging NCT among neighbors introduces communication overhead.

Yoshida *et al.* [175] proposed the Popularity-aware Dynamic Caching (PopDCN) scheme, in which the network topology is divided into multiple clusters of the same size to begin with. Each router is assigned a unique ID in the cluster for helping in content caching. Subsequently, these cluster sizes are varied (increased or decreased) based on the popularity of the contents stored in these nodes. Every piece of content chunk is mapped to a specific router within the cluster using a hash function. This ensures no content redundancy within the cluster, as each content chunk is cached at a single location determined by the hash function. PopDCN first tries to retrieve content from the originating cluster. If this fails, the request is forwarded to the nearest router in another cluster. PopDCN uses a cache advertisement approach to efficiently retrieve content, where each router updates its nearest neighbor about its cache state whenever there is a change. In this work, the authors used a grid topology for simulation, where clusters can be easily divided into equal sizes. However, in a real

network, the main challenge lies in forming clusters while keeping their sizes balanced.

Gui and Chen [50] developed an on-path content placement scheme that incorporates node and content popularity. This method considers both local and global content request patterns. Local popular content is cached at the edge router to fulfill requests from local region consumers, while globally popular content is cached at routers based on the estimated cache benefit. The cache benefit is determined by considering requests from all consumers and the node's popularity. Global popularity information is gathered by establishing communication among the edge routers within the topology. Each edge router maintains an Interest Statistical Table (IST) to track local content requests, and at the end of each cycle, edge routers exchange ISTs with each other. Additionally, all routers maintain an Information Record Table (IRT) to track the entry face and hop count of Interest packets along the route. In the downstream direction, content is cached in routers based on the *Cache Nodes* field of the Data packet. This field contains the ID of the router where the content will be cached in the downstream direction. If it has more than two router identities, caching occurs as follows: for globally popular content (beyond the dynamic threshold), it will be cached at all routers; for potentially globally popular content, it will only be cached at the first router in the returning path. If the content is globally popular and if there is no router in the *Cache Nodes* field, it is cached at the downstream router of the provider node.

The communication overhead in exchanging global popularity information among edge routers is contingent on the number of cycles. A higher number of cycles enables a more accurate collection of request patterns, but also introduces additional communication overhead. Conversely, fewer cycles reduce communication overhead but may impact performance as routers exchange data over longer intervals.

In [72], a centralized caching technique OFAM-CCN is presented. This uses the OpenFlow architecture to manage content caching. OFAM-CCN implements a centralized entity called the Cache Management (CM) table, responsible for caching popular content items within the domain. CM keeps track of the popularity table to determine which content is popular among consumers within the domain. Each time the

OpenFlow controller receives a request, it forwards it to the CM. Upon receiving the request, the CM updates the popularity table and checks its cache; if the requested content is present in its cache, it forwards both the popularity information and the content chunks to the OpenFlow controller. In a similar spirit, another study [186] proposed a centralized caching decision for ICN leveraging the SDN framework. The SDN framework provides a global view of the network, which prioritizes caching popular videos in the network while considering the perspective of global consumers. The SDN controller is responsible for making caching decisions, routing requests to the best node in the network, and tracking both global and local content popularity by receiving statistics from all nodes periodically.

The authors of [140] proposed a cache placement technique termed SDCC, which uses an SDN controller to cache popular content over time in ICN-IoT networks. They proposed two methods for managing the cache: centralized and decentralized. A centralized (single-controller) approach manages the global view of the network by placing the control logic at a central controller, while a decentralized (distributed-controller) approach reduces overhead and addresses the single point of failure. In the decentralized approach, the network is divided into multiple clusters, each managed by its own controller.

The authors of [89] proposed a feedback-based cooperative content placement technique based on the relative popularity of content from downstream nodes. This strategy uses an implicit cooperation technique, where downstream nodes pass information to upstream nodes by adding two fields to the Interest packet: *Cache-List (CL)* and *Popularity-Correction-Parameter (PCP)*. The CL field carries the list of nodes that decide to cache the content along the forwarding path between the consumer and the provider. A node is eligible to cache content if the relative popularity of the requested content is higher than that of the least popular content currently stored at the node. On the return path, this field is copied to the Data packet, and caching takes place accordingly. The *PCP* field helps convey updated popularity information to the upstream node in order to calculate the relative popularity of the content.

**Centrality-Based Caching:** Zheng *et al.* [190] proposed the BEP (Betweenness and

Edge Popularity) caching technique to enhance caching performance in NDN. BEP combines two key metrics, node betweenness centrality and content popularity, to make better caching decisions. The main goal of BEP is to place popular content at the most important routers in the network in order to achieve the maximum benefits of caching. Popularity is dynamically measured at the edge nodes (*i.e.,* routers directly connected with the consumer). Each edge router maintains a popularity table that records how often different pieces of content are requested, and this table is periodically updated to reflect the currently popular items. The betweenness centrality metric is used to measure the importance of routers. Routers with higher betweenness centrality lie on multiple shortest paths between consumers and sources, and can serve a large number of requests if they cache popular content. The calculated popularity of content is appended to the Interest packet by the edge router. As the Interest packet travels towards the source, each intermediate router adds its betweenness value to the packet. By the time the request reaches the provider (router/original source), the caching system has knowledge of both the popularity of the requested content and the importance of routers along the delivery path. In the return direction, the selected on-path router caches the content based on content popularity and router importance. This method avoids redundancy and improves the cache hit ratio.

In order to avoid caching the same content at multiple locations along the content delivery path, a hybrid content placement strategy is proposed in [102], which handles the least and most popular content differently. The less popular content is cached at edge nodes to enhance availability, while the most popular content is cached at nodes with the highest centrality.

Several works [5, 7, 166] consider content freshness along with popularity when making caching decisions. Caching fresh popular content in the routers enhances user experience and reduces server load. In this direction, Amadeo *et al.* [7] proposed a caching technique that considers content freshness along with popularity. They designed different caching approaches for core and edge routers. Core routers independently make caching decisions when content arrives, based on its popularity and freshness. As core routers handle a substantial number of requests, caching popular

and fresh content enhances the cache hit ratio. Independent caching decisions at the core reduce overhead and help make faster caching decisions. Popularity and freshness are determined by a dynamic threshold periodically estimated in the time interval `T`. If both popularity and freshness values reach the threshold, content is always cached; if the value is below this threshold, it's never cached. For popular but less fresh content, the caching decision is based on a derived probability. On the other hand, caching decisions at edge routers differ from those at the core. Caching content at edge routers promotes content diversity through limited cooperation. Through this implicit cooperation, routers set the *CACHED* flag to true in the response packet after caching content to prevent the same content from being cached by other on-path routers.

Alduayji *et al.* [5] introduce a caching scheme to bring popular content closer to the network's edge, facilitating efficient content retrieval with fewer hops. Each router manages a compact popularity table to monitor request frequencies, alleviating the computational load. Once this table is full, routers use the Least Frequently Used (LFU) policy for content eviction, prioritizing the removal of less accessed entries to accommodate new requests. Content is cached if the popularity count reaches the dynamic threshold; otherwise, caching is avoided. When router capacity is reached, the least popular content is replaced with more popular content, ensuring a cache with fresh content. This strategy results in an increase in content popularity counts near consumers and a decrease in counts for routers away from the consumers.

Wu *et al.* [166] introduced a caching technique wherein routers make content caching decisions based on dynamically estimated cache benefit values. The priority for caching is determined by a higher cache benefit, which is calculated by considering factors such as content popularity, freshness, and the hop count between the consumer and the router.

Li *et al.* [94] proposed a caching technique termed Popularity-Based Caching with Number-of-Copies Control (PB-NCC) to improve content distribution in ICN. Based on the size of the network and the popularity of particular content, copies are replicated and cached at different locations. PB-NCC considers content popularity and strictly controls how many replicas of any given content are present in the network. This

helps to achieve better content distribution and diversity, as caches are occupied with different content items. For popularity estimation, PB-NCC uses a local popularity estimation technique, where each router locally determines which contents are more frequently requested. This strategy selects at most one on-path router to cache each content item, which reduces content redundancy.

Recently, the in-network caching feature of NDN has been used in various domains such as mobile networks [183, 88], vehicular networks (VNDN) [143, 187], Internet of Things (NDN-IoT) networks [139], and satellite networks [127, 168] to improve content retrieval time. Works [9, 43, 51, 96, 153] proposed caching techniques for VNDN, considering content request preferences and node mobility to enhance content distribution and cache diversity in high-mobility environments. Wang *et al.* [153] suggested a Popularity-Incentive Caching Strategy (PICS) for VNDN to cache content in vehicle nodes, thereby reducing content delivery time and the load on the base station. In PICS, the base station acts as a leader, announcing the price of content chunks to other nodes in the network for making caching decisions. Based on the announced price, vehicles make caching decisions by considering additional factors such as location information and content popularity. To enhance content diversity, Dhara *et al.* [43] proposed a caching method, POPS-cache, where popularity estimation is done based on content categories (e.g., sports, social media, education, technology) within each region rather than the popularity of individual chunks. Amadeo *et al.* [9] introduced DANTE, a caching mechanism for VNDN that aims to improve content diversity among neighboring vehicles. In DANTE, vehicles make caching decisions based on content lifetime, popularity, and availability of content in the neighboring vehicles. Liao *et al.* [96] designed EADC, a dynamic caching approach for VNDN, to efficiently handle frequently changing content demand in vehicular networks. To determine the probability of caching the content in the vehicle, EADC takes into account multiple factors, such as vehicle characteristics (social attributes and content popularity), motion centrality (degree and betweenness centrality), and transmission cost (transmission delay, transmission difficulty, and cache redundancy).

Considering the energy-constrained environment of IoT, Zhang *et al.* [189] designed

a caching technique for ICN-IoT networks using Deep Q-Networks (DQN), which can intelligently make caching decisions by considering dynamic factors such as content popularity and lifetime. The authors of [69] designed a cooperative on-path caching technique for CCN-enabled Wireless Sensor Networks (WSNs), aiming to cache content at each sensor node. They identify the best router for caching based on its distance from the source and the degree. Another work [160] proposed a caching technique for WSNs by leveraging the in-network caching feature of NDN. Sensor nodes estimate the caching weight based on multiple factors, such as content popularity, content lifetime, availability, and resource capability, to determine which content to cache.

## 2.3 Research Gaps in Existing Literature

Many works found in the literature fall into two categories: on-path caching and off-path caching. On-path caching may operate in a cooperative or non-cooperative manner, while off-path caching is typically performed cooperatively among routers. As discussed earlier, off-path caching is more challenging than on-path caching, as it requires a significant level of coordination among routers to make caching decisions. Although several approaches have been proposed for both on-path and off-path caching in NDN, many research gaps still remain in these categories. Here, we highlight the key research gaps identified from existing works.

- Most existing studies [53, 83, 93, 118, 166, 190] focus on on-path caching schemes for content retrieval in NDN networks due to their simplicity. However, such methods do not fully exploit the potential benefits of content available in nearby off-path routers. This results in poor utilization of content available in nearby caches and increases the load on content sources when the requested content is not found in any on-path router. It may also increase retrieval time if the content available in an off-path router is closer to the consumer than the on-path router.

- Many existing caching approaches [7, 8, 19, 60, 104, 118] either use no cooperation or limited cooperation for caching decisions, which fail to utilize scarce

cache resources efficiently. As a result, the same content may be cached at multiple locations in the network, thereby reducing cache diversity and affecting the cache hit ratio because routers may experience frequent cache misses for other requested content.

- Most existing off-path approaches [62, 124, 140, 156] improve cache hit ratio and content diversity, but introduce significant communication overhead. In the literature, some off-path approaches [72, 140, 186] rely on a centralized entity (*e.g.,* SDN) to obtain a global view of the network topology. However, such a design is prone to a single point of failure and can also increase the processing and communication load on the central node. On the other hand, distributed approaches [112, 124] maintain a table to store the availability of content at neighboring routers, which requires continuous communication to keep the information updated. If the cache information becomes stale, it may increase both communication overhead and content retrieval time.

- Prior works [10, 7, 5, 19, 111, 190] lack consideration of global request patterns when evaluating content popularity, relying primarily on local request patterns at individual routers. This limitation may reduce the overall benefit to a larger number of consumers, as caching decisions are based only on local request patterns.

These research gaps indicate that there is significant scope for improvement in NDN caching. To address these limitations, we design cooperative caching techniques that leverage content available in off-path routers closer to consumers, thereby enhancing content retrieval in NDN. The proposed approaches reactively search for content in neighboring routers, avoiding the need to maintain cache state information and reducing cache misses due to stale data. Furthermore, the proposed caching approaches consider both local and global request patterns when making caching decisions, enabling more effective identification of popular content and benefiting a larger number of consumers. This, in turn, improves the cache hit ratio and content diversity across the network.

## 2.4 Summary

In this chapter, we presented an overview of existing caching techniques for ICN/NDN. We began by highlighting the importance of caching in Internet architecture. We also discussed various categories of caching techniques, such as centralized, decentralized, cooperative, non-cooperative, on-path, off-path, popularity-based caching, and caching based on network topology characteristics. We then described the workings of several well-known caching techniques and highlighted their benefits and limitations. With an understanding of how existing techniques work and their shortcomings, we present new caching algorithms in the following chapters to improve content distribution in NDN.

# Chapter 3

# eNCache: Improving Content Delivery with Cooperative Caching in Named Data Networking

## 3.1 Introduction

In-network caching is one of the most important characteristics of the NDN architecture, which allows content to be cached closer to consumers. This feature helps reduce content retrieval time and the burden on the content originator. As discussed in Chapter 2, routers can either collaborate with neighboring routers for content discovery and caching or make independent decisions [181]. A non-cooperative approach is simple to implement but results in poor utilization of cache capacity. On the other hand, a cooperative approach improves performance, but its effectiveness depends on how cooperation is implemented. Existing caching solutions for NDN use two types of approaches for content exploration and caching, depending on the position of the router between the consumer and the provider, such as on-path and off-path approaches [45]. Each approach has its own advantages and disadvantages. The on-path approach is simple and widely adopted because it does not introduce additional overhead. However, it suffers from suboptimal cache space utilization, as it overlooks

content available in routers outside the transmission path. On the other hand, the off-path approach improves cache space utilization, resulting in a higher cache hit ratio, but it introduces additional communication overhead.

To address the aforementioned challenges, in this chapter, we present a lightweight cooperative technique, `eNCache`, for content retrieval and caching. This technique aims to improve the cache hit ratio and minimize content retrieval time while adding minimal overhead. Our approach involves cooperation among neighbors in both the content search and caching phases. `eNCache` follows the NDN symmetric forwarding model, where requests and responses take the same path. In addition, it also checks for content availability in the router outside the transmission path for faster consumer responses. `eNCache` does not maintain any tables for content availability in the neighborhood. Instead, it explores neighboring caches on the fly while forwarding requests towards the provider. `eNCache` simultaneously explores content in both on-path and off-path routers, which does not introduce any extra delay due to off-path searches. For content caching, our approach follows a simple on-path technique, where routers located along the path between the provider and the consumer cache the content.

In summary, the key contributions of this chapter are as follows.

1. We describe a new content searching and caching strategy `eNCache`, which maximizes in-network cache utilization through lightweight cooperation among the routers.

2. `eNCache` being a cooperative caching method, combines the features of both on-path and off-path caching techniques to maximize the cache hit ratio, reduce content access time, and maintain low computational complexity.

3. `eNCache` avoids repeatedly querying the same set of routers by passing the list of already queried routers to the next hop.

4. We describe a variant of `eNCache` which further improves the cache hit ratio by placing the evicted content in the neighbor's cache using a hash-based method.

5. We perform simulations using large-scale ISP network topologies in the Icarus

simulator to demonstrate the working of eNCache and compare its performance with other on-path and off-path caching techniques to show its advantages.

The rest of the chapter is organized as follows: Section 3.2 reviews relevant work from the NDN caching literature that closely relates to our proposed method. Section 3.3 presents the motivation and design details of eNCache. Section 3.4 describes the simulation setup and results. Section 3.5 introduces the design of the hash-based eNCache variant and discusses its simulation outcomes. Section 3.6 outlines deployment considerations and highlights the limitations of eNCache. Finally, Section 3.7 concludes the chapter.

## 3.2 Prior Work

As discussed in Chapter 2, several on-path and off-path caching techniques have been proposed for NDN. These techniques operate in a cooperative or non-cooperative manner when searching for content in neighboring nodes and making caching decisions. In this section, we briefly review existing techniques closely related to our work and highlight their limitations.

### 3.2.1 On-path Techniques

*Leave Copy Everywhere (LCE) [84]:* This is the simplest on-path caching technique, where the router caches all passing content regardless of its popularity. While this approach does not introduce any communication overhead, its drawback is the inefficient utilization of router cache storage, resulting in a poor cache hit ratio.

*Leave Copy Down (LCD) [83]:* This technique reduces content redundancy caused by LCE by caching a single copy of the content between the provider and the consumer. LCD caches the content in the next-hop downstream router from the provider node. However, after subsequent cache hits, the same content replicates across all routers between the provider and the consumer, leading to inefficient utilization of on-path cache storage.

*Probabilistic Caching (ProbCache) [118]:* Routers along the path between the content provider and the consumer estimate the caching probability by considering cache capacity and weight parameters to make caching decisions. ProbCache reduces cache redundancy caused by LCE, but fails to use the cache space effectively.
*Dynamic Popularity-based Caching Permission (DPCP) [176]:* To enhance cache diversity, DPCP considers content popularity when caching. Once the cache space becomes full, it prioritizes storing popular content. This strategy is limited to on-path caching, which fails to utilize the availability of content outside the transmission path.

### 3.2.2 Off-path Techniques

*O2CEMF [62]:* This technique allows content to be retrieved from either on-path or off-path routers. O2CEMF builds a cache trail table that helps in fetching content from off-path routers. However, maintaining the cache trail is challenging due to the dynamic nature of the content.
*Hash-based Techniques [134]:* Hashing techniques use off-path routing and caching, where requests are directed to designated routers based on hash values, and these routers cache the content upon receiving the response. In [134], the authors designed several hash-based request routing and content placement strategies. In HR-Symmetric (HR-Symm), requests and responses follow the same path, allowing caching at the designated node. In HR-Asymmetric (HR-Asymm), request and response paths might differ. In HR-Asymm, content is delivered along the shortest path to the consumer, which may bypass the designated node. However, these techniques reduce content redundancy but may increase the time taken to retrieve content, as the designated node may be located far from the consumer.

## 3.3 `eNCache`: Design and Working Methodology

This section describes the working of `eNCache`, a proposed neighborhood cooperative caching technique. We begin by presenting the design rationale of `eNCache`, followed by details of its Interest and Data packet forwarding mechanism using a

reference network topology shown in Figure 3.1.

### 3.3.1 Design Rationale

The response time for content delivery in an NDN architecture improves due to the caching capabilities of routers. Usually, in the upstream direction, routers forward the Interest to the next hop without cooperation. In the downstream direction, routers independently decide which content to store and which to replace when the cache occupancy is full. This approach simplifies cache management and reduces overhead. However, this strategy ignores the availability of content in neighbors, which may lead to reduced caching performance. For example, in Figure 3.1, Consumer3 requests content `Name`, which is provided by Source6, and a copy of the content is available in the cache of router $R_9$. In this case, the request for `Name` is forwarded to Source6 using the shortest path via routers $R_3$, $R_4$, $R_5$, and $R_6$ instead of router $R_9$, which is just two hops away from Consumer3. To take advantage of the content available in nearby off-path routers, `eNCache` enables cooperation among routers in the neighborhood during both Interest and Data packet forwarding. This cooperation of `eNCache` maximizes the cache hit ratio and diversity while reducing content retrieval time and minimizing overhead associated with cooperation.

`Cooperative Caching:` Let $R_1, R_2, \cdots R_n$ be the routers in the network. Let $C_1, C_2, \cdots C_n$ be their respective caching capacities. We formally describe the working of `eNCache` with the help of the following definitions.

**Definition 1.** ***N-hop-Neighborhood($R_i$):*** *For a router $R_i$ its N-hop neighborhood includes all those routers which can be reached within $N$ hops i.e., $R_j \in$ Neighborhood($R_i$) if $D(R_i, R_j) \leq N$. Here, $N$ represents the number of hops, and $D$ is a function measuring the distance in hops.*

In the Figure 3.1, the 1-hop and 2-hop neighborhoods of router $R_{10}$ are highlighted.

**Definition 2.** ***NextHop($R_i$):*** *$R_j$ is the next hop router of $R_i$ if $R_j$ is next in the sequence of on-path routes from the requester to the service node.*

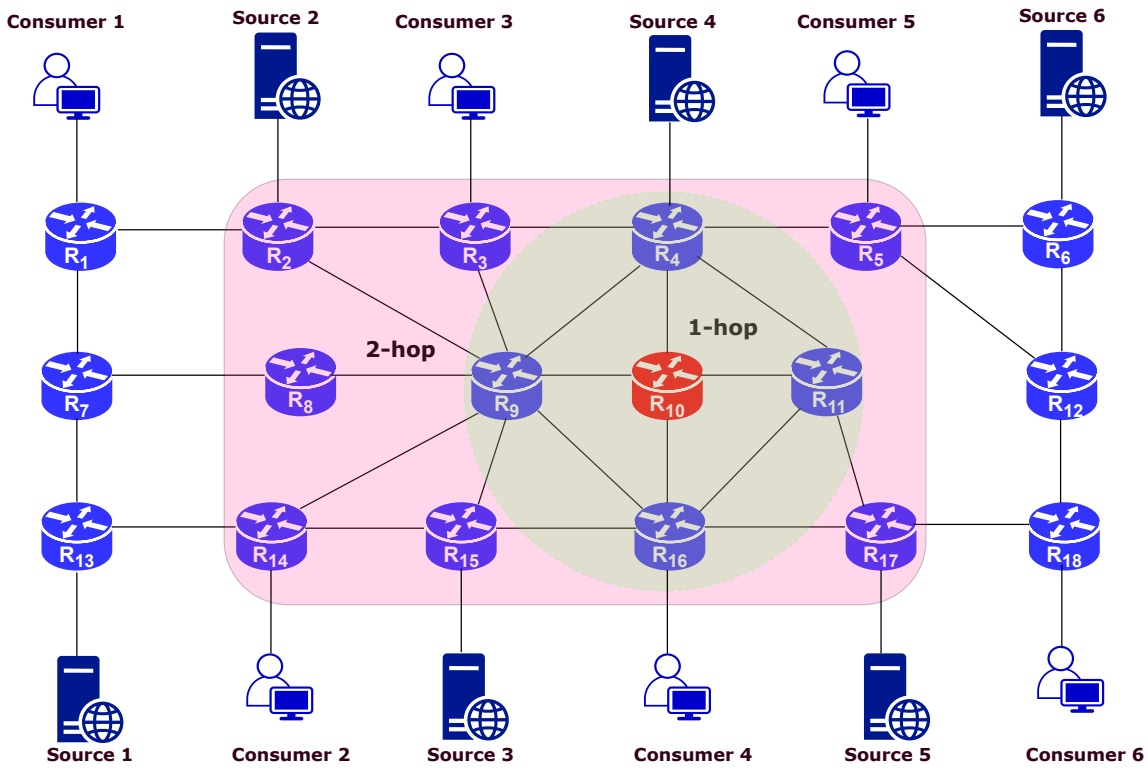



Figure 3.1: eNCache Reference Architecture.

**Definition 3.** ***N-hop-Offpath-Neighborhood*($R_i$)*:*** *For a router $R_i$, its N-hop offpath neighborhood includes all those routers which can be reached within $N$ hops, excluding the router $R_k$ which is the next on-path router i.e., $R_j \in OffPathNeighborhood(R_i)$ if $D(R_i, R_j) \leq N$ and $R_j \neq NextHop(R_i)$.*

**Definition 4.** ***Edge Router:*** *A router $R_i$ is an edge router with respect to a content requester node $n_j$; if $D(n_j, R_i) = 1$*

**Definition 5.** ***Occupancy*($R_i$)*:*** *Occupancy ($O_i$) is the sum of the sizes of all the content in the cache of router $R_i$.*

The eNCache caching technique, like other caching techniques, has two major operations:

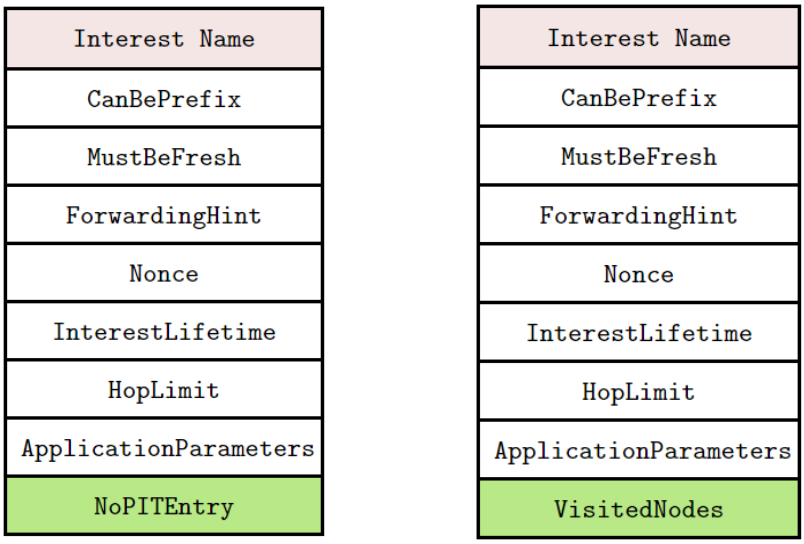
(i) Interest Forwarding (Request Routing): A content request from a consumer is directed to the nearest available source to ensure the content is retrieved in the shortest possible time (Algorithm 3.1).

(ii) Data Forwarding (Caching Decision): When the content arrives, the router decides whether to cache it. If the cache is full, it determines which content to evict in order to accommodate the new content (Algorithm 3.2).

### 3.3.2 Interest Forwarding

When a consumer requests content, the request is forwarded towards the potential provider (router/original publisher) using the shortest path forwarding technique. Algorithm 3.1 outlines how `eNCache` routes the request to the nearest provider. The consumer-generated request first arrives at the edge router (first hop router from the consumer), which checks its cache. If the requested content is found in the cache, it is served back to the consumer; otherwise, the request is forwarded further to retrieve the content. The request is forwarded not only to the next-hop on-path router but also to *N-hop* off-path neighbors. Requests to both on-path and *N-hop* off-path neighbors are sent simultaneously to increase the chances of retrieving the content quickly. To achieve simultaneous forwarding, `eNCache` uses different Interest packets for on-path and off-path requests. We have modified the NDN native Interest packet structure [3] by adding additional fields to optimize the searching process. For off-path *N-hop* neighbors, the request packet is updated with the fields *HopLimit* and *NoPITEntry*. The modified Interest packet structure for off-path is shown in Figure 3.2a. *HopLimit* $= N$ is used to restrict the off-path neighborhood search, and *NoPITEntry* = *True* is used to avoid PIT entry creation at multiple off-path routers. This ensures that off-path *N-hop* neighbors do not store requests in their PIT table. Instead forward the requests to their neighbors by reducing *HopLimit* by 1 and remembering it in auxiliary storage. For example, in Figure 3.1, if Consumer3 sends an Interest packet towards the Source2 via routers $R_3$ and $R_2$. When the on-path router $R_3$ receives the packet from Consumer3, it sets *HopLimit* = *1* and *NoPITEntry* = *True*. This means that $R_3$ forwards the packet only to its one-hop off-path neighbour(s). In this case, $R_4$ and $R_9$ are off-path routers, so they receive the packet. These off-path routers neither forward the packet further nor create PIT entries. However, for the on-path router, we update the Interest packet structure with the list of already visited routers

as shown in Figure 3.2b. It is also worth noting that when the request is forwarded to the next hop on-path router, the PIT entries are created as usual. This process is repeated at each intermediate on-path router located on the shortest path between a content requester and a content provider until the content is delivered by any of the router caches or eventually served by the content source.




(a) Off-path Interest (b) On-path Interest

Figure 3.2: `eNCache` Interest Packet Structure: *NoPITEntry* and *VisitedNodes* Fields Added to NDN Packet Format v0.3.

Table 3.1: FIB of Router $R_5$

| **Name Prefix** | **NextHop** |
|---|---|
| xyz/media/Name | $R_4$ |

Table 3.2: PIT of Router $R_5$

| **Name Prefix** | **Interface** |
|---|---|
| xyz/media/Name | $I_5$ |

Consider the topology shown in Figure 3.1 with *HopLimit* = *1* (*i.e.*, cache lookup is performed within a one-hop neighborhood). Suppose Consumer5 issues a request

**Algorithm 3.1** Processing Content Request at a Router $R_i$

1: `Name` $\leftarrow$ New Content Request at $R_i$
2: **if** $R_i$ is an Onpath Router **then**
3:  **if** `Name` $\in$ CacheOf($R_i$) **then**
4:  Serve from $R_i$
5:  **end if**
6: **else if** `Name` $\in$ CacheOf($R_j$), where $R_j \in$ *N-hopNeighborhood*($R_i$) $-Visited$ **then**
7:  Serve from $R_j$
8: **else**
9:  Visited $\leftarrow$ Visited $\cup$ *N-hopNeighborhood*($R_i$) $\cup \{R_i\}$
10:  $R_j \leftarrow NextHop(R_i)$
11:  Forward Content Request to $R_j$
12: **end if**

for the content `xyz/media/Name`, which is produced by Source2. FIB and PIT of the edge router $R_5$ are shown in Tables 3.1 and 3.2, respectively. First, the request arrives at edge router $R_5$. If there is a cache miss at $R_5$, then $R_5$ simultaneously searches for the content with both its on-path next-hop neighbor $R_4$ and its 1-hop off-path neighbors $R_6$ and $R_{12}$. Note that $R_5$ forwards the request to the next on-path router $R_4$, by consulting its FIB, as shown in Table 3.1. If the requested content is available in any of these *1-hop* routers, it will be served from that router; otherwise, the request

**Function** *N-hopNeighborhood*($R_i$)

1: $NH(R_i) \leftarrow \emptyset$
2: **for all** $R_j \in R$ and $R_j \neq R_i$ **do**
3:  **if** $D(R_i, R_j) \leq N$ **then**
4:  $NH(R_i) \leftarrow NH(R_i) \cup \{R_j\}$
5:  **end if**
6: **end for**
7: $NH(R_i) \leftarrow NH(R_i) - \{NextHop(R_i)\}$
8: **return** $NH(R_i)$

will be forwarded towards the original content source. Once the request is forwarded, an interface entry in the PIT for the content is also created, as shown in Table 3.2. Similar operations are repeated at $R_4$ and the subsequent hops until the request gets served either by any of the on-path or 1-hop off-path routers, or in the absence of cache, the content is eventually served by the content source. We note that the same router can be a neighbor for many of the on-path routers, and it would be wasteful to repeat the search for such common neighboring routers. Hence, we avoid redundant off-path searches by subsequent routers if any of the previous on-path routers have already queried these off-path routers. For example, in Figure 3.1, if $R_4$ has queried off-path router $R_9$, then the next hop on-path router $R_3$ need not query $R_9$ again. In order to achieve this, every on-path router updates a list of queried routers in the Interest packet header *VisitedNodes* (a global set), using which the next-hop on-path routers avoid redundant queries to such routers. This field enables faster lookup of content in off-path neighbors and reduces the number of messages required to explore those neighbors.

Referring to Figure 3.1, suppose Consumer3 generates an Interest for content `Name` that is served by Source6. Router $R_3$ is the first router to receive the Interest from Consumer3. Assume that the content is searched within 1-hop neighbors. $R_3$ explores its 1-hop neighbors: $R_2$, $R_4$, and $R_9$. While forwarding the Interest to the next on-path router $R_4$, it adds $\{R_2, R_3, R_4, R_9\}$ to the *VisitedNodes* field, indicating that these routers have already been explored. Router $R_4$ has 1-hop neighbors $\{R_3, R_5, R_9, R_{10}, R_{11}\}$. It does not send the Interest to $R_3$ and $R_9$, since they are already listed in the *VisitedNodes* field. After its exploration, while sending requests to its off-path neighbors, the updated *VisitedNodes* list is also attached to the Interest packet forwarded to the next on-path router. The updated *VisitedNodes* list becomes $\{R_2, R_3, R_4, R_5, R_9, R_{10}, R_{11}\}$, which is then forwarded to the next on-path router $R_5$. Router $R_5$ has 1-hop neighbors $\{R_4, R_6, R_{12}\}$. It explores $R_6$ and $R_{12}$ (excluding already visited router $R_4$) and updates the field to $\{R_2, R_3, R_4, R_5, R_6, R_9, R_{10}, R_{11}, R_{12}\}$. When $R_6$ receives the Interest, it refers to the updated *VisitedNodes* field and avoids forwarding the packet to its one-hop neighbors $R_5$ and $R_{12}$. Instead, $R_6$ directly for-

wards the Interest toward Source6. By using this field at each hop, the number of messages required to explore off-path neighbors is reduced.

### 3.3.3 Caching Decisions

When a consumer requests content, the request is routed to the nearest location of the content based on the entry in FIB. Once found, the content is sent in the reverse direction by the serving node (on-path/off-path router/original source). In addition to forwarding, each router on the path decides whether or not to cache content. `eNCache` uses the on-path caching strategy to cache the contents along the delivery path between the serving node and the requester node. We adopted the on-path caching approach because it is simple yet effective. Algorithm 3.2 summarizes the caching operation of `eNCache`. A deviation from the traditional on-path caching mechanism is that the decision to cache the content is made taking into account whether the neighborhood routers have the content. This is in order to make the best use of the routers' limited cache capacity. Let's say that new content `Name` arrives at router $R_i$. The following caching decisions take place based on the availability of space and the type of router (edge or non-edge).

If the router $R_i$ is an edge router w.r.t. the requester node, it caches the content `Name`. This is because caching the content at the edge reduces network traffic and latency. If the edge router has space, it caches the newly arrived content; otherwise, it makes room using a replacement policy to cache the new content. On the other hand, if $R_i$ is not an edge router and its cache is not full, it caches the content `Name` even if neighboring nodes already have it, to make the best use of the cache capacity. If $R_i$ is a non-edge router and its cache is full, it coordinates with its neighbors to make a caching decision. $R_i$ does not cache the content `Name` if it is already in the CS of a neighbor; otherwise, it caches it in its CS by removing an existing content according to the cache replacement policy. This cooperative approach helps maintain diverse content within the neighborhood and increases the chances of serving future requests.

For example, in the topology of Figure 3.1, suppose the new content `Name` arrives at $R_2$ and if $R_2$ has spare capacity, $R_2$ caches it; otherwise, $R_2$ consults its neighbor

routers $R_1, R_3, R_9$. When none of the neighboring routers have content `Name`, $R_2$ caches it by replacing the existing entry.

**Algorithm 3.2** Caching Content at Router $R_i$

```
1: Name ← New Content at R_i
2: Size ← SizeOf(Name)
3: if R_i is an Edge Router then
4:    Cache Name in R_i
5: else if Name ∉ CacheOf(R_j ∈ Neighborhood(R_i)) then
6:    Cache Name in R_i          /* Use cache replacement policy */
7: else
8:    if C_i − O_i ≥ Size then
9:       Cache Name in R_i       /* Maximize cache utilization */
10:   end if
11: end if
```

### 3.3.4 Communication Overhead Analysis of `eNCache`

The overhead of retrieving content from a provider located *N-hops* away from the source consists of two components.

(i) Searching the *N-hop* neighborhood for the content: This we denote as *N-hop* Neighborhood Search Delay (NSRD) and includes the time for sending requests to the neighborhood and receiving the response from them. Assuming uniform delay of $T_N$ between all nodes this turns out to be twice the distance between querying node $R_i$ and a node $R_j$ at $N$ hops away. If the delay is not uniform, then $T_N$ assumes the maximum of all delay values.

*N-hop* Neighborhood Search Delay (NSRD) $= 2 \times N \times T_N$.

For example, consider router $R_3$ in Figure 3.1. If $R_3$ needs content that is not locally cached, it initiates an $N$-hop search (e.g., 1-hop, 2-hop, etc.). Assume that the delay between any two adjacent routers is 1 ms ($T_N$). For a 1-hop search, NSRD (request + reply) is: $\text{NSRD}_{(\text{1-hop})} = 2 \times 1 \times 1 = 2$ ms. Similarly, for a 2-hop search, $\text{NSRD}_{(\text{2-hop})} =$

$2 \times 2 \times 1 = 4$ ms.

If links between routers have varying delays, for example $(R_3, R_2)$ : 3 ms, $(R_3, R_4)$ : 5 ms, and $(R_3, R_9)$ : 7 ms, then $T_N$ is taken as the maximum delay (7 ms) to represent the worst-case scenario. Hence, for a 1-hop search: $\text{NSRD}_{\text{(1-hop)}} = 2 \times 1 \times 7 = 14$ ms.

(ii) Forwarding the request along the on-path: The Interest packet is also simultaneously forwarded to the next on-path router, as it is also part of *N-hop* neighborhood. This router will initiate similar lookup and search operations as that of the previous router. However, it excludes the neighborhood routers of previous hops. This does not alter the delay incurred (asymptotic complexity). Thus, forwarding the Interest packet to the next-hop neighborhood does not take extra time. This set of querying and forwarding operations is performed up to N–1 times in the worst case. Thus, the total delay incurred in the forward direction is $(N-1) \times T_N$.

For example, consider router $R_3$ in Figure 3.1. Assume that the delay between adjacent routers is 1 ms ($T_N$). When $R_3$ forwards the Interest packet towards Source6, it is sent to the next on-path router $R_4$. Router $R_4$ performs a similar lookup, excluding the routers already queried by $R_3$. In the worst case, the Interest packet is forwarded along the path $R_3 \rightarrow R_4 \rightarrow R_5 \rightarrow R_6$. Hence, the total delay in the forward direction is: $(4-1) \times 1 = 3$ ms.

(iii) Sending the content back to the requester: This in the worst case takes $N-1$ forwards. Thus the total delay is $(N-1) \times T_N$.

Once the content is found at $R_6$, it is sent back along the same path ($R_6 \rightarrow R_5 \rightarrow R_4 \rightarrow R_3$). In the worst case, this also involves $(N-1)$ hops. Hence, the return delay is: $(4-1) \times 1 = 3$ ms.

Thus, the Content Retrieval Delay (CRD) can be estimated as in Equation 3.1.

$$CRD = (N-1) \times 2 \times N \times T_N + 2 \times (N-1) \times T_N \tag{3.1}$$

Referring to Figure 3.1, consider router $R_3$ forwarding an Interest towards Source6. For the 1-hop search scenario with a link delay of $T_N = 1$ ms between adjacent routers, the overall CRD is: $3 \times 2 \times 1 \times 1 + 2 \times 3 \times 1 = 12$ ms.

## 3.4 Performance Evaluation

In this section, we describe the simulation setup, network topology, and evaluation metrics used for performance assessment, followed by a discussion of the results.

### 3.4.1 Simulation Setup

We use Icarus [135], a widely adopted Python-based ICN/NDN cache simulator. Icarus is specifically designed for testing caching strategies in ICN/NDN environments. It provides a modular framework to facilitate realistic large-scale simulation with customized topologies and trace-driven traffic workload. It also allows us to create custom trace files with the specified request patterns. Our experiment workload consists of two files: i) a content trace — which contains all unique contents (*i.e.,* the content catalog), and ii) a request trace — which includes all the consumers' access patterns of content requests. In our simulations, the content trace contains 1000 unique content objects in the network, while the request trace contains 100,000 requests made by the consumers. An initial 10,000 requests were generated for cache warm-up to ensure the cache reaches a steady state before collecting performance metrics. The source nodes (content generators) receive a uniform distribution of content objects. For example, if the topology has ten source nodes, each source node contains 100 content objects. Consumers request content following a Poisson distribution model with a mean rate of $\lambda = 1$ request per second. In this model, most requests occur around the mean arrival time, with fewer requests dispersed towards the tail of the distribution. Hence, this distribution determines the order and number of times each consumer made the Interest request during the simulation. In the workload that we created, consumers request the content objects in a round-robin fashion, *i.e.,* all the unique contents have been queried. As we have 1000 unique content names, any consumer will query for content after all 1000 contents have been requested. We consider round-robin request order mainly to explore how various caching strategies behave when each content has an equal chance of being requested by the consumers and when each content has equal popularity, so that each content has the same chance of being cached by the

routers. The round-robin request order represents the worst-case scenario, and any other request order will result in better performance.

For simplicity, in our simulation, we assume that each router holds the same amount of content, and each content item has the same size. The term $C$ in Table 3.3 represents the total network cache capacity as a proportion of the total content population accessible in the network. All caching strategies employ the Least Recently Used (LRU) eviction policy, which replaces the least recently used content with new content when cache slots are full. Caching techniques use Dijkstra's shortest path algorithm to route the Interest and Data packets across the network. Each experiment is conducted ten times to reduce experimental error. For all plots, we report the average of ten runs along with 95% confidence intervals. During the simulation, all caching strategies use the same experimental parameters, as detailed in Table 3.3.

Table 3.3: Simulation Parameter Settings

| **Parameters** | **Values** |
|---|---|
| Network topology | RocketFuel ISP topologies |
| Number of content objects | $10^3$ objects |
| Number of content requests | $10^5$ objects |
| Cache warm-up | $10^4$ objects |
| Content request order | Round robin |
| Request rate | 1 request per second |
| Cache size | $C \in [1\%, 3\%, 5\%, 7\%, 10\%]$ |
| Initial cache capacity | Empty |
| Cache replacement policy | LRU |
| Content placement | Uniform distribution among sources |
| Workload | Trace-driven |
| Experiment repetitions | 10 |

### 3.4.2 Network Topology Setup

We conducted simulation experiments on two large-scale RocketFuel Internet Service Provider (ISP) network topologies [146, 147], AS 1221 (Telstra, Australia) and

Table 3.4: Details of Network Topologies

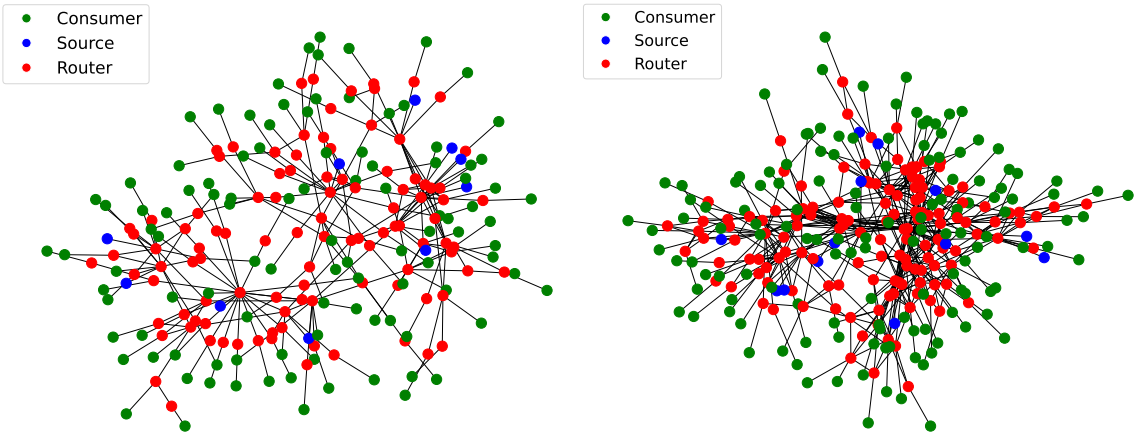
| Topology Name | Node | Edge | Consumer | Source | Router |
|---|---|---|---|---|---|
| AS 1221 Telstra (Australia) | 218 | 265 | 104 | 10 | 104 |
| AS 6461 Abovenet (US) | 289 | 523 | 138 | 13 | 138 |

AS 6461 (Abovenet, US). These topologies differ in size and region. They are generated using the Fast Network Simulation Setup (FNSS) toolchain [133] within the Icarus simulation environment [135]. The RocketFuel topology dataset includes detailed information on routers, links, bandwidth, and latencies, which were collected from different regions of ISP maps. To enable caching, the ICN stack is installed on each router node. The consumer and source nodes (original provider) are artificially created and connected to the routers. These newly added nodes have a degree of 1 and are connected to routers with a degree greater than 1. In our simulation experiments, we used the default RocketFuel topology settings provided by the Icarus simulator [135]. The consumer nodes are assigned to each router in the parsed topology (*i.e.*, the total number of routers in the topology equals the number of consumer nodes), and the ratio between the number of source nodes and routers is set as 0.1. The link delay between the consumer-to-router and the source-to-router has been set to 0ms (this can be any other value), while the link delay for the router-to-router links is parsed from the RocketFuel latency dataset [135]. Figure 3.3 shows the ISP maps of the AS 1221 and AS 6461 topologies, where consumer nodes are shown in green, source nodes in blue, and router nodes in red. Table 3.4 summarizes the details of these network topologies.

### 3.4.3 Benchmark Schemes and Evaluation Metrics

We compare the performance of `eNCache` with state-of-the-art caching strategies: LCE [84], LCD [83], ProbCache [118], and DPCP [176]. The details of these benchmark schemes are presented in Section 3.2. `eNCache` is evaluated against these strategies using four commonly used metrics in the NDN caching literature.

*1. Cache Hit Ratio:* It measures the proportion of content requests served by the




(a) AS 1221 (b) AS 6461

Figure 3.3: RocketFuel ISP Topologies: AS 1221 and AS 6461.

routers from their caches instead of directly from the content source. Higher the cache hit ratio, better the caching strategy utilizes the limited capacity of routers for content placement. The overall Cache Hit Ratio (denotes as $\overline{\mathrm{C}}$) is calculated using Equation (3.2).

$$\overline{\mathrm{C}} = \frac{\sum_{i=1}^{N} R_{\mathrm{hit}_i}}{N} \tag{3.2}$$

where $N$ denotes the total number of requested content objects in the network, and $R_{\mathrm{hit}_i}$ indicates the fraction of content served from the router cache for the $i^{th}$ request. Referring to Figure 3.1, cache hit ratio in the simulation is measured as follows. Suppose Consumer1 generates 10 content requests. Out of these, contents `Name1` and `Name2` are served from the cache of $R_1$, `Name3` and `Name4` from the cache of $R_2$, and `Name5` from the cache of $R_7$. The remaining five requests are served directly from Source1 and Source2. In this scenario, 5 out of 10 requests are served from cached content within the network, resulting in a cache hit ratio of 0.5.

*2. Content Access Time (Latency):* This measures the total time from when the consumer makes a request to when the content is delivered to them. A lower delay indicates faster content delivery to the consumer. The overall delay (denotes as $\overline{\mathrm{D}}$) between content request and response is determined using Equation (3.3).

$$\overline{\mathrm{D}} = \frac{\sum_{i=1}^{N} \mathrm{IPD}_i + \sum_{i=1}^{N} \mathrm{DPD}_i}{N} \tag{3.3}$$

where $IPD_i$ denotes the time taken by the Interest packet to reach the serving node from the consumer, and $DPD_i$ denotes the time taken by the Data packet to return from the serving node to the consumer. $N$ is the total number of requests made by the consumer during the simulation.

Referring to Figure 3.1, let Consumer1 make one request and Consumer3 make one request. Consumer1 requests `Name1`, which is served from Source1. Assuming the link delay of each interface is 1 ms, IPD is 4 ms and DPD is also 4 ms. Consumer3 requests `Name2`, which is served by router $R_4$, resulting in an IPD of 2 ms and a DPD of 2 ms. Therefore, the overall average content access time for these two requests served to the consumers is 6 ms.

*3. Cache Diversity:* It measures the total number of unique contents stored collectively across all routers in the network. For example, if content `Name1` is cached in 3 routers and content `Name2` in 2 routers, the cache diversity is 2, as there are two distinct contents cached across the network. Cache diversity $\overline{D}$ ranges from 0 to $M$, where $M$ denotes the total number of distinct contents in the network. A value of $\overline{D} = 0$ indicates that no content is cached in any router in the network, whereas $\overline{D} = M$ implies that all available contents are cached at least once across the network. A caching strategy with high cache diversity indicates better overall content availability in the network.

*4. Average Hit Distance:* It measures the average number of hops taken by Data packets from the content serving node to the content requester node. It is denoted as $\overline{\mathrm{H}}$. The lower the value of $\overline{\mathrm{H}}$, the more number of content objects are served by nearby routers. It is calculated using Equation (3.4).

$$\overline{\mathrm{H}} = \frac{\sum_{i=1}^{N} Dist(serv_i, req_i)}{N} \tag{3.4}$$

where $serv_i$ represents the content serving node, $req_i$ represents the requester node, $Dist(serv_i, req_i)$ represents the distance (hop count) between $serv_i$ and $req_i$, and $N$ represents the total number of requested contents.

For example, in Figure 3.1, if Consumer1 requests `Name1`, which is served by $R_6$, the hit distance is 6. Similarly, if Consumer3 requests `Name2`, which is served by $R_4$,

the hit distance is 2. Therefore, the average hit distance in this case is 4.

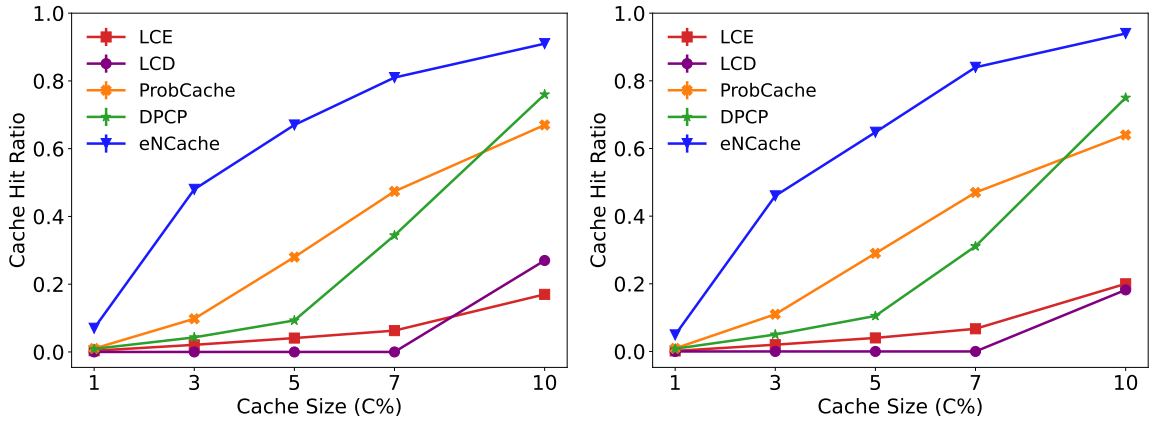
### 3.4.4 Simulation Results and Observations

This subsection presents a comparative analysis of the proposed `eNCache` and four benchmark techniques, evaluated on two different network topologies and varying simulation parameters.

***Impact of Varying the Cache Sizes:*** It is worth noting that, cache size is an important consideration and can affect the performance. For example, the cache hit ratio will be affected if the cache size is smaller, as more requests will be served by the content producer. Similarly, other parameters like latency, diversity, and hit distance are also affected by the cache size. Thus in order to understand the impact of different cache sizes, we compare the performance of `eNCache` with other methods for different cache sizes. For this performance comparison, we varied the cache sizes as a fraction of the total number of content objects. The performance of all caching strategies is evaluated using the metrics described previously, with each node's cache size (C) set to 1% to 10% of the total content population available in the network.

Figure 3.4 illustrates the effect of cache size on the cache hit ratio. All caching strategies benefit from larger network cache sizes, resulting in higher hit ratios. Compared to on-path schemes, the hit ratio of the proposed `eNCache` cooperative scheme is always higher for any cache size. `eNCache` achieves a cache hit ratio of approximately 90% on both ISP network topologies when the network cache size is set to 10%. This signifies that cooperative content caching decisions effectively utilize network cache space by reducing content redundancy. ProbCache and DPCP outperform other on-path strategies, such as LCE and LCD, because these methods make caching decisions based on the cache weight factor. LCE and LCD performance is poor due to high content redundancy and poor utilization of available cache on the delivery path.

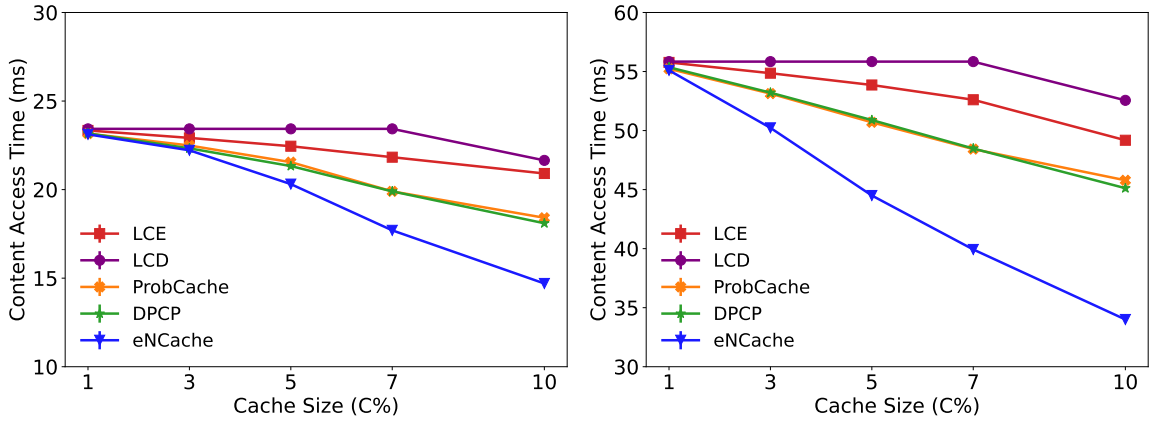
We also compared the content access latency (content retrieval time) by varying the cache size. Figure 3.5 depicts the content retrieval time for various caching methods over different cache sizes. We can see that `eNCache` has the lowest content retrieval time compared to other caching strategies across all cache sizes. This improvement in




(a) AS 1221 (b) AS 6461

Figure 3.4: Cache Hit Ratio for Different Cache Sizes on Different Network Topologies.

content access latency is primarily due to `eNCache's` off-path content discovery and the cooperative caching decisions made by the routers. Furthermore, we can observe that as the cache size increases, the content access time for all methods decreases. This trend is due to the fact that more content is cached in the network routers.

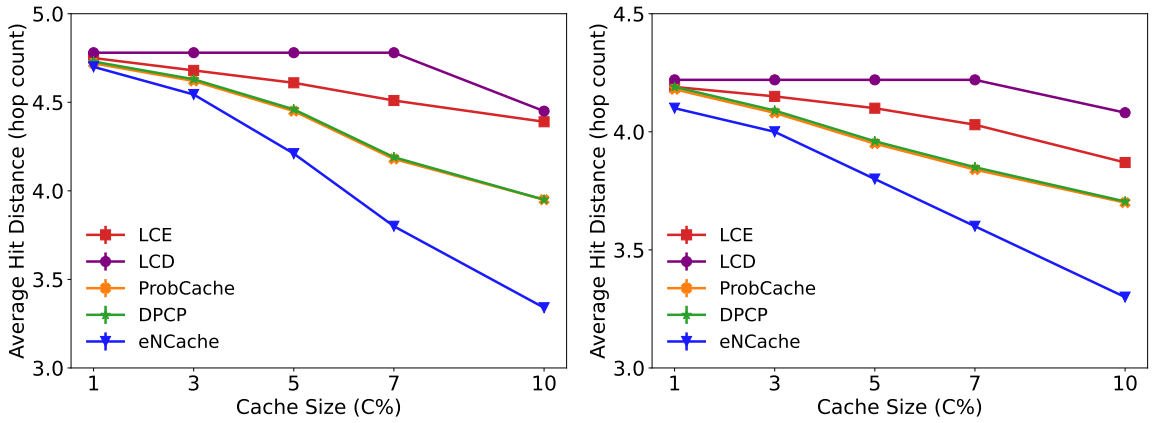



(a) AS 1221 (b) AS 6461

Figure 3.5: Content Access Latency for Different Cache Sizes on Different Network Topologies.

Figure 3.6 depicts the average hit distance of caching strategies in terms of hop count over the various cache sizes. From Figure 3.6, we can observe that all caching strategies significantly reduce the hops taken by Data packets to satisfy user requests

as the cache size increases. This trend is because nearby routers serve the requested content. The proposed strategy outperforms other caching strategies regardless of the network cache size due to its efficient cooperative content discovery and cache placement. The reduction in hop distance signifies faster content delivery and a lesser load on the actual content source.

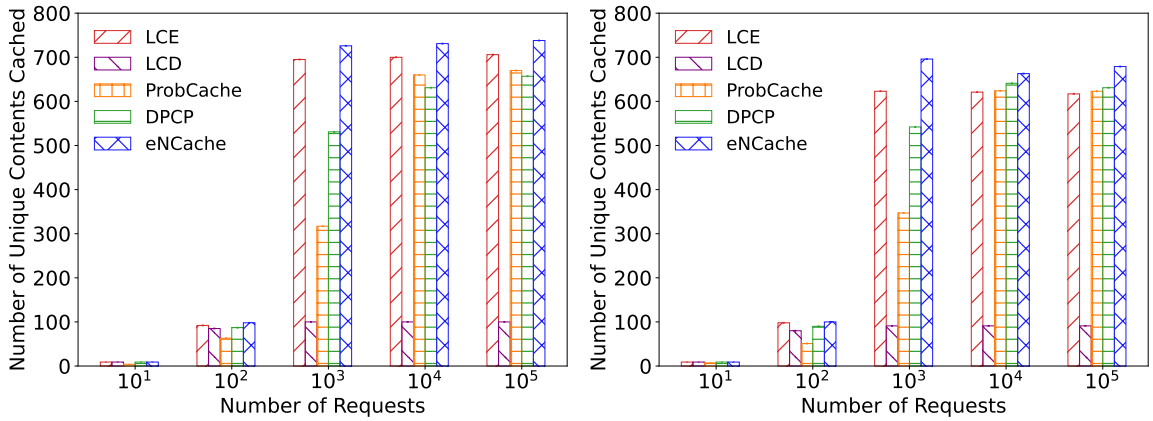



(a) AS 1221 (b) AS 6461

Figure 3.6: Average Hit Distance for Different Cache Sizes on Different Network Topologies.

***Impact of Varying the Number of Requests:*** Due to the limited cache capacity, cache diversity changes over time due to content eviction. To capture this behavior, we measured cache diversity with respect to the number of consumer-generated requests. To assess the effectiveness of different caching strategies, we configured the total network cache capacity to $C = 1\%$. A smaller cache size allows us to better observe how efficiently each strategy retains unique content across the network.

Figure 3.7 shows the number of unique contents cached (cache diversity) in the network for all caching strategies across both network topologies. From the results shown in Figure 3.7, we can observe that the proposed `eNCache` strategy has the highest number of distinct contents cached across all scenarios. Higher cache diversity demonstrates the fundamental reason for the proposed strategy's superior performance in terms of the higher cache hit ratio and lower content access cost, even with a smaller cache size.



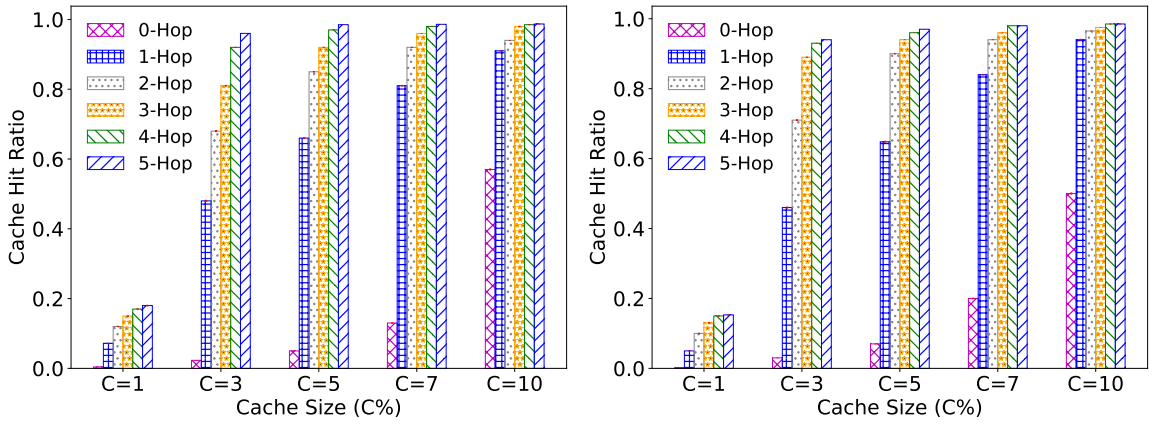

(a) AS 1221 (b) AS 6461

Figure 3.7: Cache Diversity: Comparison of the Unique Contents Cached in the Network for Different Network Topologies (C=1%).

***Impact of Varying the Neighborhood Distance:*** It is worth noting that `eNCache` queries the neighborhood nodes/routers for content and also forwards the requests to the next on-path router. A natural question here is: What is the size of this neighborhood? More number of off-path routers in the neighborhood; better the chance that content being found and served. However, this may introduce additional overheads. For example, consider a scenario where the requested content is not found in any of the routers and ultimately gets served by the source (producer) node. In this case, `eNCache` will flood requests at every node along its *N-hop* unique neighbors, adding to the additional forwarding path delay of $(N-1) \times T_N$. Also, the request packet size keeps increasing at each node due to the addition of $N$ nodes in the *VisitedNodes* at each hop. Thus, a very large neighborhood size of $N$ may increase the worst-case latency. Moreover, a neighborhood size beyond a certain limit may offset the gains achieved by `eNCache`. To understand reasonable values of $N$, we study the performance of `eNCache` by varying the neighborhood sizes.

We vary the off-path neighborhood cache lookup from *1-hop* to *5-hop* and *0-hop* for no off-path neighborhood cache lookup on different cache sizes to see how it affects `eNCache` performance in terms of the cache hit ratio and time to retrieve the requested contents.

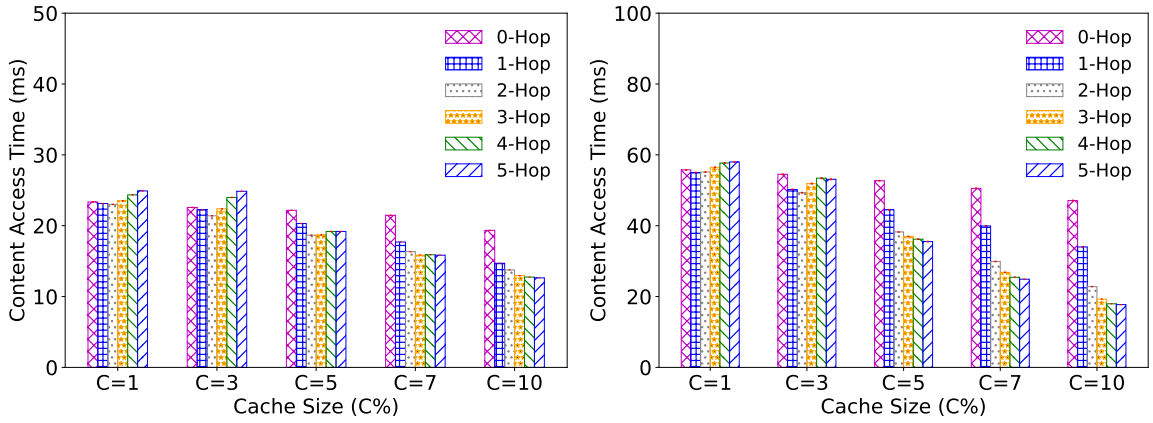
Figure 3.8 depicts the overall cache hit ratio for different network topologies on different cache sizes. We can notice from Figure 3.8 that the cache hit ratio increases as the number of hops for the off-path cache lookup increases. This trend is due to the fact that nearby off-path routers serve a large number of contents without going to the source node, as each on-path router is connected with many neighbors in the larger network topology. Furthermore, as the routers' cache slots increase, more content is retrieved from off-path routers. Increasing the *HopLimit* field value improves the cache hit ratio and reduces the load on the source node, but at the same time increases communication overhead. Increasing the *HopLimit* beyond a certain point offers little benefit, as the cache hit ratios for *4-hop* and *5-hop* show only a marginal difference. Hence, the extent of off-path exploration should be configured according to the network topology size.




(a) AS 1221 (b) AS 6461

Figure 3.8: Cache Hit Ratio: `eNCache` Performance on Different Cache Sizes with Varying Hop Counts (*0-hop* to *5-hop*).

Figure 3.9 depicts the content access latency for our proposed caching strategy when the HopLimit parameter varies from *0-hop* to *5-hop* across both the ISP topologies. From Figure 3.9, we can observe that at network cache capacity $C = 1\%$, until *2-hop* off-path cache lookup, we get better performance in terms of access latency by retrieving content from nearby off-path nodes before going to the original content source (*0-hop*) across all topologies. This trend changes as the value of C increases




(a) AS 1221 (b) AS 6461

Figure 3.9: Content Access Time: `eNCache` Performance on Different Cache Sizes with Varying Hop Counts (*0-hop* to *5-hop*).

because increased cache capacity means requested contents are available in nearby cache nodes. Two points are worth noting: first, increasing the *HopLimit* parameter causes communication overhead, and second, real network nodes have limited storage space to store the contents. Thus, increasing the value of the *HopLimit* parameter does not offer much benefit after a certain extent.

## 3.5 Hash based `eNCache`

`eNCache` is designed to always cache the requested content at the edge router, irrespective of whether there is available space or not. When the space is not available, based on the cache replacement policy, the edge router evicts an existing item from its cache to make room for this newly arrived content. Now, we present a modified version of `eNCache` which gives a second chance for the content being replaced. The idea is to make use of the available cache in neighboring routers by placing evicted content into one of the caches of a neighboring router. This makes `eNCache` an off-path caching approach, as content is stored in a neighbor that lies outside the shortest path between the provider and the consumer.

Algorithm 3.3 shows the procedure for relocating the content from the edge router

to the other neighboring router. The operation of the edge router is modified, the subroutine `CacheAtEdgeRouter()` performs the caching of the current content at the edge router and the caching of the evicted content at the appropriate neighborhood router. First, it checks for the available cache size and places the new content at the edge router if it has sufficient capacity. Otherwise, it evicts the existing content from its cache to make space for the new content and calculates the hash $h$ — a numeric value derived by passing the name of the evicted content to a function $f()$. This $h$ is used to identify the corresponding neighbor router where the evicted content is stored by using the mod operation as shown in step 9 of the subroutine. For example, consider the reference topology shown in Figure 3.1. $R_3$ is an edge router for Consumer3. When a new content `Name` arrives at $R_3$ and the cache at $R_3$ cannot accommodate it, the old content `OName` is evicted to make room for `Name`, and the evicted content `OName` is cached in one of $R_3$'s neighboring routers (*i.e.,* $R_2$, $R_9$, or $R_4$). Assume that after performing the mod operation at $R_3$, the designated neighbor is identified as $R_9$ to cache the evicted content; then the evicted content is forwarded to $R_9$ for caching and will be cached if space is available at $R_9$; otherwise, `eNCache` ignores caching of the evicted content.

Hashing-based `eNCache` differs from `eNCache` with respect to how the caching decisions are made at the edge router when all of the edge routers' cache is occupied. In such cases, the edge router provides a second chance to the evicted content and tries to cache it at another router within the neighborhood. Thus, the overhead is only incurred at the edge router, wherein for every new request after the edge router's cache is full, there is an additional message generated to cache the replaced content in the neighbourhood router. For example, if the cache space of edge router $R_i$ is full, it sends a suggestion for the evicted object (`OName`) to its neighbor $R_K$. This results in an additional message being generated at the edge router.

### 3.5.1 Simulation Results of Hash-based `eNCache`

Here, we compare the hash-based `eNCache` with off-path content searching and caching techniques outlined in Section 3.2, such as O2CEMF [62] and two popular

**Algorithm 3.3** Caching Content at Router $R_i$ Using a Hash-Based Technique

```
1: Name ← New Content at R_i
2: Size ← SizeOf(Name)
3: if R_i is an Edge Router then
4:    CacheAtEdgeRouter(R_i, Name)        /* Subroutine call */
5: else if Name ∉ CacheOf(R_j), where R_j ∈ Neighborhood(R_i) then
6:    Cache Name in R_i        /* Use cache replacement policy */
7: else
8:    if C_i − O_i ≥ Size then
9:       Cache Name in R_i        /* Maximize cache utilization */
10:   end if
11: end if
```

**Function** *CacheAtEdgeRouter*($R_i$, Name)

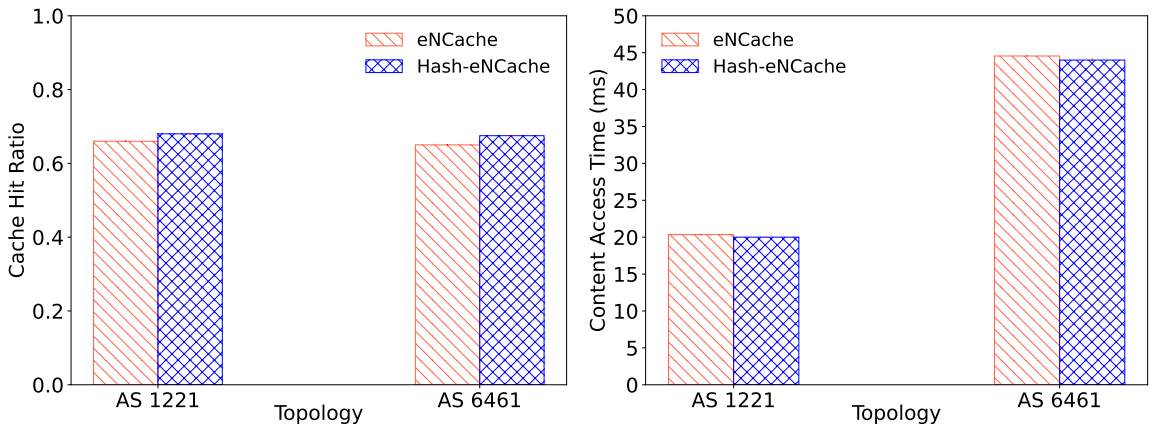
```
1: Size ← SizeOf(Name)
2: if C_i − O_i ≥ Size then
3:    Cache Name at R_i
4:    O_i ← O_i+ Size
5: else
6:    OName ← EvictCache(R_i)
7:    NeighborSize ← SizeOf(Neighborhood(R_i))
8:    h ← f(OName)
9:    K ← hmod NeighborSize
10:   if C_i − O_i ≥ SizeOf(OName) and OName ∉ CacheOf(R_K) then
11:      Cache content OName at R_K
12:   end if
13: end if
```

hash-based off-path approaches: HR-Symm and HR-Asymm [134]. In HR-Symm, content requests and replies follow the same path (in opposite directions), whereas in HR-Asymm, the paths may be asymmetric. The O2CEMF strategy explores content

in neighboring off-path routers using a controlled flooding technique during Interest forwarding. It operates in two phases: exploration, where routers discover off-path neighbors and build cache trail tables based on Data or NACK responses; and exploitation, where this information guides the forwarding of future requests. Although O2CEMF also employs network coding and rank-based block matching, we focus solely on its cache exploration and routing mechanisms for performance comparison, excluding the network coding component. O2CEMF uses the LCE approach for caching, where each router along the response path caches the content. The `HopLimit` (a field controlling Interest forwarding) is configured to 1 in both the `eNCache` and O2CEMF strategies to explore content within 1 hop neighbors.

Here, we use the same simulation setup, network topologies, and evaluation metrics to compare Hash-based `eNCache` with the benchmark off-path techniques, as described in Section 3.4.

First, we compare the performance of both versions, `eNCache` and hash-based `eNCache`, using two key metrics: cache hit ratio and content access time. Figure 3.10 shows the results on both topologies with a cache size of 5%. The simulation results show that the hash-based `eNCache` outperforms the standard `eNCache`, achieving a higher cache hit ratio and lower content access time on both topologies.



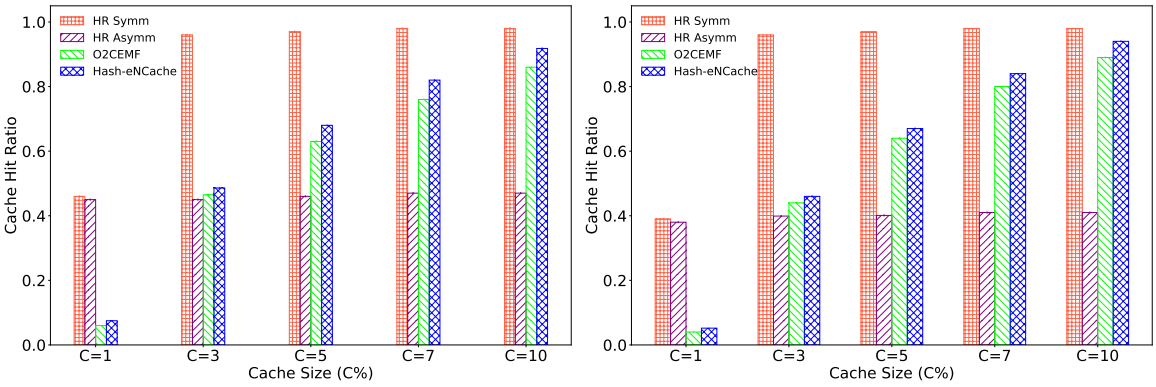

(a) Cache Hit Ratio

(b) Content Access Time

Figure 3.10: Performance comparison of `eNCache` with Hash-based `eNCache` on Different Network Topologies (C=5%).

***Impact of Varying the Cache Sizes:*** Here, we present the simulation results of all four techniques, highlighting the impact of varying cache sizes on cache hit ratio, content access time, and hop distance.

Figure 3.11 shows the impact of cache size (varied between 1% to 10%) on the cache hit ratio. From Figure 3.11, we can observe that the hash routing strategy HR-Symm outperforms O2CEMF and `eNCache` in terms of cache hit ratio for cache sizes 1% to 7%, but, as the cache size increases to 10%, the cache hit ratio of O2CEMF and `eNCache` tends to be closer to that of HR-Symm. Additionally, for cache sizes above 3%, the cache hit ratio of O2CEMF and `eNCache` surpasses that of HR-Asymm on both topologies. This difference is due to HR-Symm always caching content at the designated node, whereas HR-Asymm caches content only if the node is on the shortest path. Additionally, hash routing ensures content is cached at just one router in the network, reducing redundancy to 0%. While this may seem a reasonable choice, we demonstrate that it can adversely affect the content access latency, as the designated router may be far-off from the consumers. `eNCache` consistently achieves a higher cache hit ratio than the O2CEMF, regardless of cache sizes and topologies. This improvement can be attributed to the cooperative caching decisions made by routers in `eNCache`.

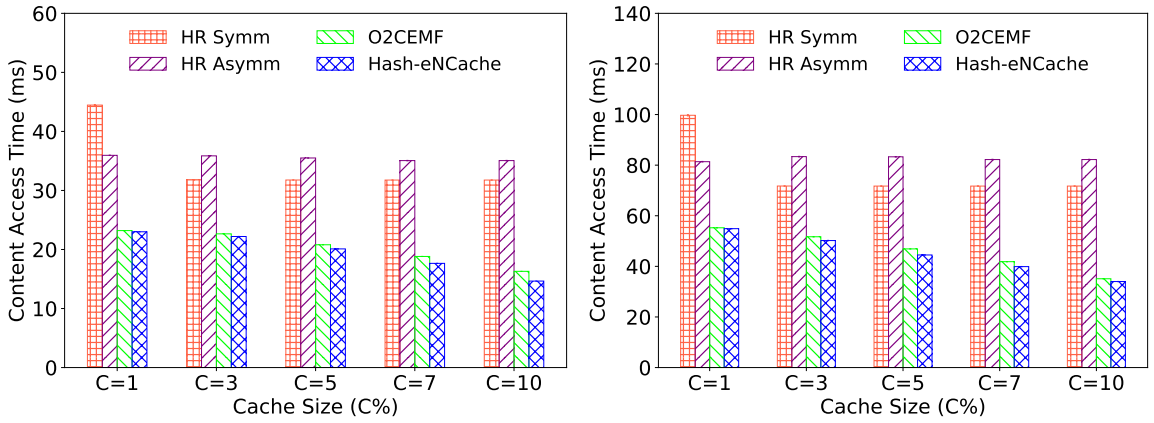



(a) AS 1221 (b) AS 6461

Figure 3.11: Cache Hit Ratio Comparison of Hash-based `eNCache` with Varying Cache Sizes for Different Network Topologies.

Figure 3.12 illustrates the results of content access latency achieved with four

caching strategies for different cache sizes. Compared to the other two hash routing strategies (HR Symm and HR Asymm), O2CEMF and our proposed `eNCache` strategies have lower latency. Unlike the two hashing strategies that try to fetch the content from a designated node, O2CEMF and `eNCache` fetch the content from the nearest on-path or off-path router by exploring content in the neighboring routers while forwarding the request toward the original content source. Thus, `eNCache` and O2CEMF result in better latency compared to hash-based techniques, as in large networks, the designated cache node can be too far from the requester, leading to increased retrieval time. Furthermore, the latency of `eNCache` is slightly lower than O2CEMF. This can be attributed to a better collaborative caching mechanism of `eNCache`, which allows routers to cache a diverse range of content closer to the consumers.

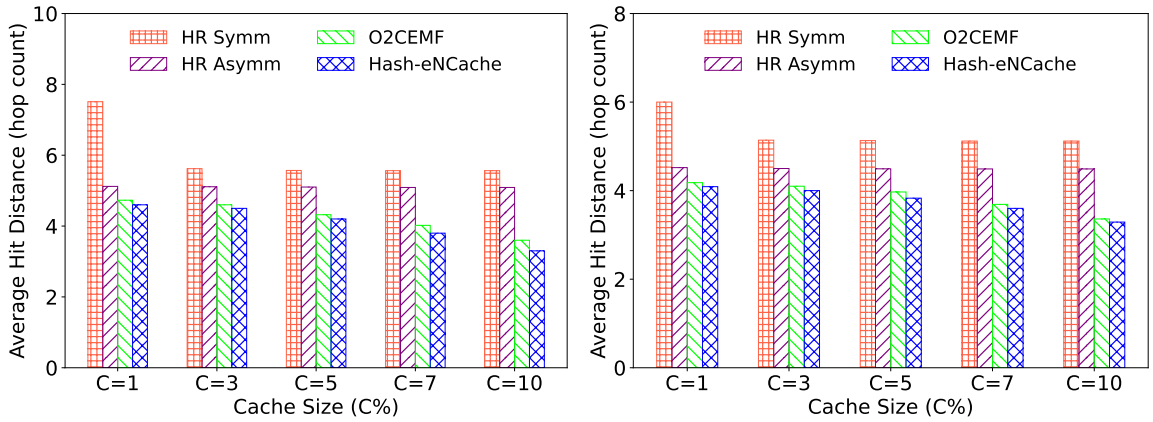



(a) AS 1221 (b) AS 6461

Figure 3.12: Content Access Time Comparison of Hash-based `eNCache` with Varying Cache Sizes for Different Network Topologies.

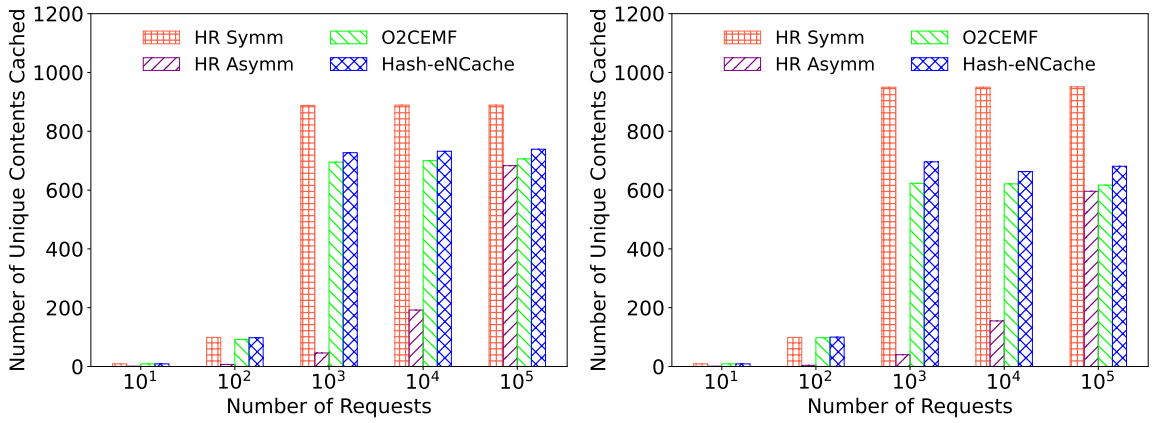
Figure 3.13 illustrates the average hit distance of all four caching strategies in terms of the number of hops the Data packet has traveled in the network over the various cache sizes. The results show that the `eNCache` strategy achieves a maximum of up to 46% fewer hops than the HR Symm technique, up to 35% fewer hops than the HR Asymm technique, and up to 9% fewer hops than the O2CEMF technique. The higher value of the average hit distance of hash-based techniques clearly indicates that the designated cache node is located far away from the requesters. This metric

demonstrates why eNCache has lower content access latency than the other two hash-based (HR Symm and HR Asymm) techniques and O2CEMF.




(a) AS 1221 (b) AS 6461

Figure 3.13: Comparison of Average Hit Distance for Hash-Based eNCache Across Varying Cache Sizes and Network Topologies.




(a) AS 1221 (b) AS 6461

Figure 3.14: Cache Diversity Comparison of Hash-Based eNCache Across Different Network Topologies (C=1%).

***Impact of Varying the Number of Requests:*** Similar to the measurement of cache diversity in eNCache, we evaluated the hash-based eNCache by varying the number of requests from $10^1$ to $10^5$. Figure 3.14 shows the cache diversity metrics

for four caching methods (HR Symm, HR Asymm, O2CEMF, and eNCache), *i.e.,* the number of unique contents available across all router caches in the network with a cache size of 1%. As the number of requests increases, the number of unique contents cached by the router also grows because it receives more requests and caches a more diverse set of content. However, due to limited cache capacity, content diversity fluctuates as some content is evicted to make room for new content. From Figure 3.14, it can be observed that the cache diversity of the O2CEMF and eNCache is lower than HR Symm because the O2CEMF and eNCache caching techniques may end up caching the same content at multiple routers in the network to reduce content access latency. However, due to the cooperative nature of the eNCache strategy, it achieves higher cache diversity than O2CEMF across both topologies.

### 3.5.2 Signaling Overhead

It is worth noting that eNCache, being a cooperative caching technique, introduces additional overhead due to Interest messages sent to a few off-path routers. In order to measure the signaling overhead, we conducted a comparative analysis of HR Symm, HR Asymm, O2CEMF, and hash-based eNCache strategies in terms of the total Interest message exchanged to fetch the content. O2CEMF also generates NACK packets to update the cache trail table when content is not found. For consistency in comparison with other strategies, we omitted the overhead from NACK packets in this simulation. Simulations on both AS 1221 and AS 6461 topologies were performed using the same setup and parameters as listed in Table 3.3.

Figure 3.15 shows the number of Interest messages exchanged for exploring content in the neighboring off-path routers (*Off-path HopLimit=1*) for O2CEMF and hash-based eNCache strategies and in the designated routers for hash-based HR Symm and HR Asymm strategies. From Figure 3.15, we can observe that the O2CEMF and hash-based eNCache strategies involve a higher number of Interest message exchanges compared to the hash-based HR Symm and HR Asymm strategies. This is due to the fact that O2CEMF and hash-based eNCache strategies search for content in neighboring routers through Interest flooding during request forwarding, whereas the HR

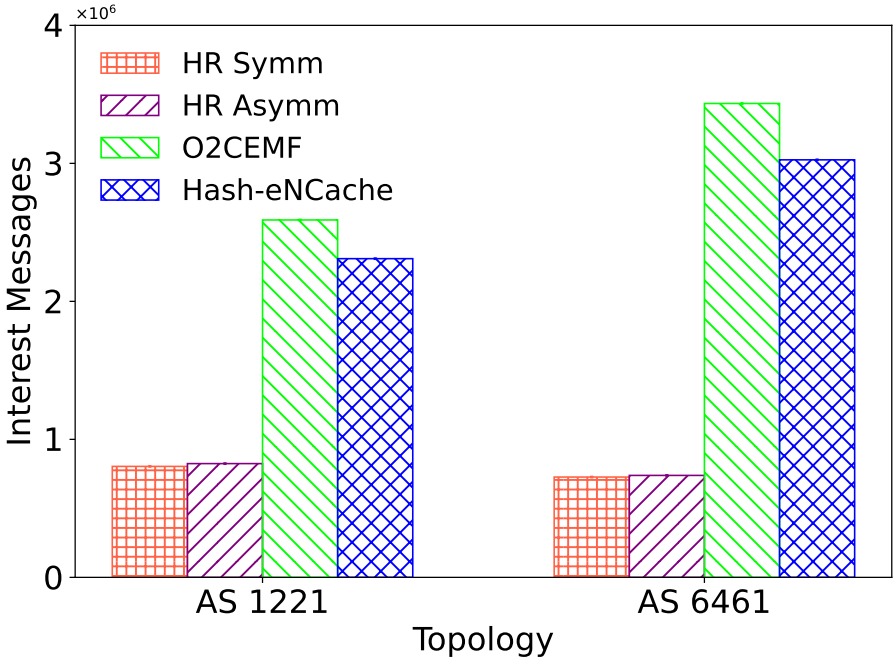



Figure 3.15: Comparative Analysis of Total Interest Message Exchanges Across Different Network Topologies.

Symm and HR Asymm strategies directly query the designated router to locate the content without the need for flooding Interests. Furthermore, Figure 3.15 illustrates the advantage of incorporating the *VisitedNodes* field in the Interest packet structure of hash-based `eNCache` when sending requests to the content source. This design choice reduces the need for packet flooding in the `eNCache` strategy compared to the O2CEMF strategy for off-path content exploration. This reduction is due to the fact that, in a large network topology, many nodes share common neighborhoods, so passing the information about previously explored routers to the next-hop on-path router will effectively avoid redundant searches.

We also conducted simulations to examine how many additional messages are exchanged in hash-based `eNCache` when the *VisitedNodes* field is disabled. Figure 3.16 compares the number of messages exchanged to explore on-path and off-path neighbors in hash-based `eNCache` with the *VisitedNodes* field enabled versus disabled. When the *VisitedNodes* field is enabled, hash-based `eNCache` requires 13.7% fewer messages exchanged compared to when the *VisitedNodes* field is disabled on the AS 1221 topology,

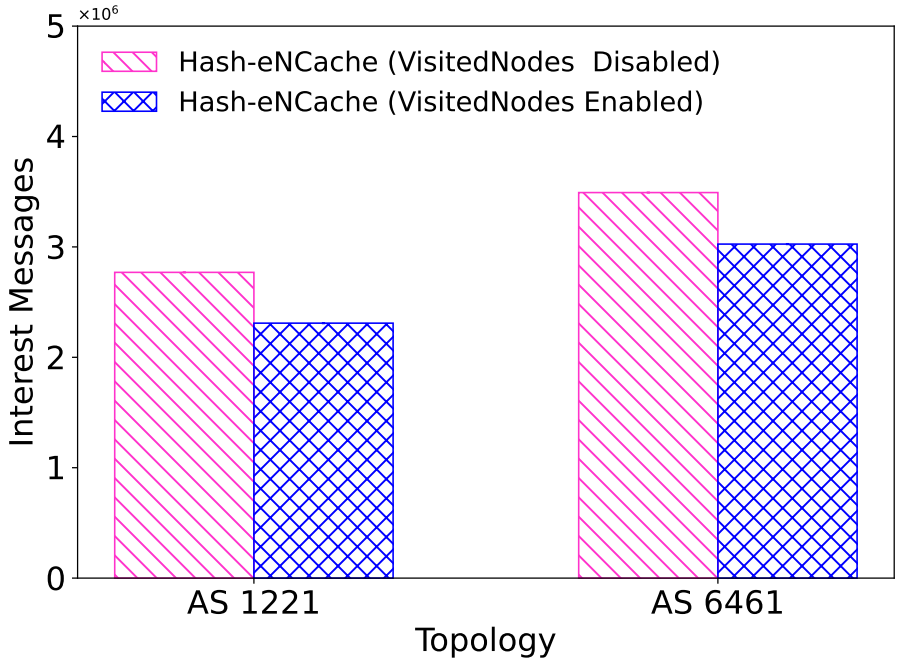



Figure 3.16: Comparative Analysis of Total Interest Message Exchanges in `eNCache` with *VisitedNodes* Field Enabled vs. Disabled.

and 13.9% fewer on the AS 6461 topology. As the topology becomes denser (*i.e.,* nodes share many common neighbors) and consumers generate more requests, the difference in messages exchanged may increase further.

## 3.6 Discussion

In this section, we discuss practical considerations of `eNCache`, particularly the implications of appending *VisitedNodes* to the Interest packet, the possibility of receiving duplicate packets, and packet losses.

*Interest Packet Overheads:* `eNCache` strategy includes a *VisitedNodes* field in Interest packets to prevent revisiting previously explored routers. On-path routers located between the requester and provider first extract and analyze data from the *VisitedNodes* field. Subsequently, they append the information about visited routers before forwarding the packet to the next-hop on-path router. This prevents the generation of duplicate Interest packets at each router and hence avoids duplicate requests to the off-path routers. However, this field introduces additional overheads on both packet

data and computation (packet processing) for the on-path routers. The overhead is directly related to the size of the *VisitedNodes*. Let $X$ be the set of neighbors visited so far up to the previous hop of router $R_i$ and $Y$ be the set of neighbors of router $R_i$, then the number of neighbors that need to be visited by $R_i$ is $Y$-$X$. Let the set of common nodes between $X$ and $Y$ be $Z$. So here, processing overhead at each on-path router is incurred while determining the set $Z$. This is finding the overlap between the two sets. Once router $R_i$ has determined $Z$, it appends $Y$-$X$ to the set $X$ and forwards this updated set to the next upstream router. Hence, the size of set $X$ continues to grow at each successive hop.

*Duplicate Interest Packets:* It is possible that a router $R_j$ receives two Interest packets from two different routers for the same content. For example, refer to the Figure 3.1. Let us assume that a content is requested by Consumer5, which is produced by Source5. The content request follows the shortest path through routers $R_5$, $R_{12}$, $R_{18}$, and $R_{17}$. When a content request arrives at router $R_5$ and needs to be forwarded to its 2-hop neighbors, router $R_{12}$ may receive requests for the same content from both $R_6$, a 2-hop neighbor and $R_5$, a router on the path. However, in this scenario, $R_5$ obtains the content from $R_{12}$ (if available) and sends it back to the consumer without waiting for a response from $R_6$. $R_6$ may also forward the Data to $R_5$, but this reply is ignored since content requests are forwarded simultaneously to on-path and off-path routers. This does not create any operational issues in the `eNCache`.

*Packet Loss:* It is worth noting that an Interest packet sent to an on-path or off-path neighbor may be lost and may not reach some or all of the neighbors. Also, the Data packets sent by neighbors may be lost as well. However, this is not a very serious problem. All on-path losses are handled by the NDN default loss handling method, while any off-path (Interest or Data) packet loss will be treated as the neighbor not having the content, and the content will be fetched from other on-path routers or the producer itself. For example, in Figure 3.1, when Consumer1 requests the content `Name`, which is provided by Source4. The request follows the shortest path through routers $R_1$, $R_2$, $R_3$, and $R_4$. The off-path query limit is set to *1-hop*. Assuming copies of the content `Name` are available in the caches of $R_7$ and $R_3$. $R_1$ simultaneously for-

wards requests towards Source4 and *1-hop* off-path router $R_7$. If the content response forwarded by $R_7$ is lost due to congestion or link failure, $R_1$ retrieves the content `Name` from on-path router $R_3$ and sends it back to Consumer1 without waiting for a response from $R_7$. This simultaneous forwarding does not add any additional querying delay caused by querying off-path neighbors.

## 3.7 Summary

In this chapter, we presented `eNCache` to improve the overall performance of content delivery. This improvement is measured with the cache hit ratio, content access time, cache diversity, and hit distance. Efficiency of content delivery and changed operations of `eNCache` are facilitated with modifications done to the NDN Interest packet. These modifications enable off-path searching and prevent requests from being forwarded to routers that have already been visited by previous hops. We compared `eNCache` with well-known on-path and off-path caching techniques from the NDN caching literature. Simulation results demonstrate the effectiveness of `eNCache`, owing to its cooperative content querying and caching approach. However, `eNCache` caches all content regardless of its popularity, which leads to frequent content replacement in the cache. To address this limitation, the next chapter introduces a popularity based cooperative content searching and caching technique that caches content based on its popularity.

# Chapter 4

# PeNCache: Popularity based Cooperative Content Searching and Caching Technique for NDN

## 4.1 Introduction

The cooperative content searching and caching technique, eNCache, presented in the previous chapter, demonstrates that cooperative decision-making (fresh caching decisions and search operations) among neighbors improves cache utilization and brings efficacy for content retrieval. However, eNCache's caching decisions are deterministic and popularity-agnostic, which could result in the eviction of popular items from the cache. A way to improve eNCache's performance is to take the popularity of content into account, along with neighborhood cooperation. Several studies [7, 10, 19, 166] suggested that caching frequently requested contents in the network improves the performance of the caching system. In order to distinguish between popular and non-popular content, routers need to maintain a table to store the frequency of content. As discussed in Chapter 2, content popularity can be measured either at the local level, where each router considers how many times it has seen that content, or at the global level, where the demand of all consumers is taken into ac-

count. Global popularity is beneficial, as many routers are part of multiple content delivery paths. However, estimating global popularity requires the exchange of messages among routers, which increases the overhead in the network. We aim to design a caching method that takes into account the popularity of content while exhibiting reasonable overhead. We incorporate both local and global popularity into account. For global popularity estimation, we use a lightweight cooperation approach.

In summary, the key contributions of this chapter are as follows:

1. We propose `PeNCache`, a lightweight cooperative content searching and caching method for NDN to make better use of in-network cache storage.

2. `PeNCache` considers both local (popular within a specific router) and global (popular across the entire network) content popularity while making caching decisions to maximize the possibility of serving frequently accessed content directly from the nearest cache.

3. For the estimation of global popularity, `PeNCache` selects a few designated nodes (leaders) based on the diameter of the network. Each edge node is linked to a designated node for exchanging content popularity information.

4. The performance of `PeNCache` is evaluated using the Icarus simulator and compared against representative NDN caching strategies on large-scale network topologies using standard evaluation metrics.

5. We also conduct an analysis of communication overhead to evaluate the effectiveness of `PeNCache` and discuss the challenges associated with deployment.

The rest of the chapter is organized as follows: Section 4.2 reviews relevant work from the NDN caching literature that is closely related to our proposed method. Section 4.3 describes the motivation and design details of `PeNCache`. Section 4.4 presents the simulation setup and results. Section 4.5 discusses deployment considerations and the shortcomings of `PeNCache`. Finally, Section 4.6 summarizes the chapter.

## 4.2 Prior Work

As discussed in Chapter 2, several popularity-agnostic and popularity-based caching techniques have been proposed for NDN. These techniques can be cooperative or non-cooperative in nature. In this section, we briefly review existing techniques closely related to our work and highlight their limitations.

### 4.2.1 Popularity-agnostic Caching

Some caching approaches, like Leave Copy Everywhere (LCE) [84], are simple, where every node along the delivery path caches the content. However, this leads to content redundancy and inefficient cache utilization. To address this, other approaches like Prob(p) [181], ProbCache [118], and Max-Node Utility (MNU) [75] adopt techniques to find the suitable places to cache the content. Prob(p) and ProbCache are probability-based caching techniques in which routers cache content based on a probability value. In Prob(p), a router caches the content with a fixed probability value $p \in [0, 1]$. Unlike Prob(p), ProbCache dynamically estimates the probability value by considering factors such as the node's distance along the content delivery path and cache weight parameters. In MNU, the most suitable router(s) along the delivery path are selected to cache the content. The selection is based on a utility value, which is computed using both the router's topological position and its dynamic attributes.

### 4.2.2 Popularity-based Caching

Popularity-based caching techniques make caching decisions based on the frequency of content requests. Most-Popular Content (MPC) [19] caches an item only when its request count at a particular router exceeds a predefined threshold. However, this strategy fails to utilize cache space efficiently. Along with content popularity, Popularity-aware Closeness-based Caching (PaCC) [8] also considers the router's proximity to the majority of consumers when making caching decisions. Another caching technique, Priority-based Content Popularity-Aware (PCPA) [101], enhances the cache hit ratio by considering both content popularity and router significance in the caching deci-

sion. In order to improve cache space utilization, Caching-Resource-Utilization-Based Strategy (CRUS) [93], proposed by Li *et al.*, identifies on-path routers with the lowest resource utilization to cache the content. However, these strategies do not utilize content stored in routers outside the transmission path.

To leverage the benefits of content available in caches outside the transmission path, authors of [124] proposed an Adaptive Threshold-Based Cooperative Caching Method (ATCM). In this strategy, routers dynamically calculate a threshold based on content popularity, distance, and replacement rate to make caching decisions. Each router maintains a Neighbor Cache Table (NCT), which records the content available in its *1-hop* neighbors. This table helps retrieve content from nearby off-path routers. However, keeping the NCT up to date requires frequent cache information exchanges among *1-hop* neighbors, which significantly increases communication overhead.

Caching approaches that do not make coordinated caching decisions often fail to utilize cache space efficiently, whereas cooperative approaches achieve better cache utilization but at the cost of increased communication overhead.

## 4.3 PeNCache: Design and Working Methodology

This section describes the working details of `PeNCache`. We begin by presenting the motivation behind `PeNCache`, followed by a description of the system model. Next, we describe the operation of routing requests, popularity estimation, and caching decisions using the reference network topology shown in Figure 4.1.

### 4.3.1 Motivation

As discussed earlier, popularity-based caching techniques are beneficial in improving cache hit rates and reducing content delivery time. Due to the limited cache capacity of routers, accounting for content popularity has become a key factor in improving caching performance in NDN [10, 19, 101]. The objective of `PeNCache` is to combine the benefits of cooperative caching while accounting for content popularity to improve content availability within the network and minimize content delivery time.

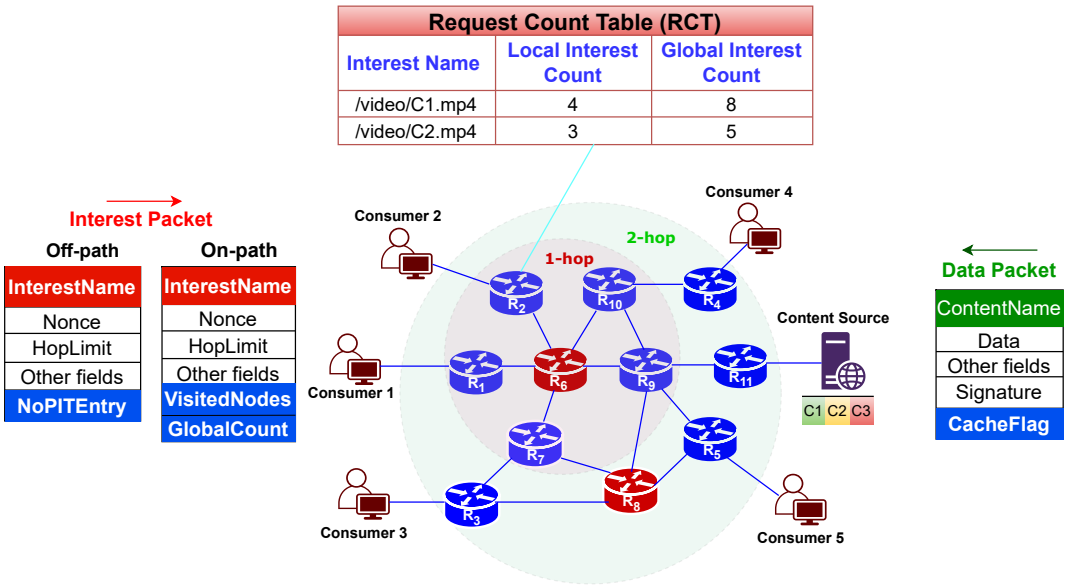



Figure 4.1: PeNCache Reference Architecture.

### 4.3.2 System Model

We present the working of PeNCache with reference to a topology shown in Figure 4.1. This reference network has five consumers, one content source, and 11 routers ($R_1$ to $R_{11}$) with caching capabilities. Among these routers, five are edge routers ($R_1$ to $R_5$), which are directly connected to consumers, and the remaining six are non-edge routers. Routers $R_6$ and $R_8$ serve as the designated nodes (also referred as leader nodes) for collecting the global popularity of the content. Edge nodes interact directly with the nearest designated node to exchange the request count of the contents. In addition to the essential NDN data structures (CS, PIT, and FIB), each router in the proposed design maintains an additional data structure termed the Request Count Table (RCT), like router $R_2$ in Figure 4.1, to track the frequency of requests. This table is used for local popularity estimation. RCT contains three fields: the name of the request, the count of the request at the individual router, and the count of the request across the network.

In order to facilitate cooperative content search and also to take popularity into account, we modified the NDN Interest and Data packet structure [3] with the addition

of the following fields. Fields 1 to 4 are added to the Interest packet, while Field 5 is added to the Data packet.

1. *HopLimit:* An integer in the set of natural numbers that specifies how many hops off the main path the search operation can go. On-path routers send the query message up to the *N-hop* off-path routers.

2. *VisitedNodes:* A set of router identities visited by the on-path router(s). Each on-path router adds information about visited neighbors in this field to prevent redundant exploration of common neighbors by other routers along the route. The *VisitedNodes* field is not added to the Interest packet when it is forwarded to off-path neighbor(s).

3. *NoPITEntry:* A one-bit flag indicating whether a router should create an entry for this content request. When sending the query to the *N-hop* off-path routers, the on-path router sets the *NoPITEntry* field to True in order to prevent the off-path routers from waiting for the specific content. The *NoPITEntry* flag is not set in the Interest forwarded to the next-hop on-path router, because on-path routers maintain the PIT state of the forwarded Interest. This PIT state assists in routing the content back to consumers along the reverse path and in making caching decisions.

4. *GlobalCount:* A numerical value that indicates the number of times a particular content is requested in the network. The edge router adds the global count of the request into this field after obtaining it from the designated node. This value is utilized by the routers located on the content delivery path to measure the popularity of content while making caching decisions. This field prohibits other routers from interacting with the designated node to get the global count of the requested content. The *GlobalCount* field carries a value when the Interest is forwarded to on-path routers and is set to empty when the Interest is forwarded to off-path neighbors.

5. *CacheFlag:* A one-bit flag set to either True or False that helps make caching decisions. This flag is added in the Data packet to minimize content redundancy among directly connected neighbors. The *CacheFlag* field carries the router ID and cache status. When a router forwards a Data packet towards requested consumers, it adds its ID and cache status as True (if the requested content is already available or cached) or False (if the content is not cached) so that the next router along the

transmission path makes an informed caching decision.

We use the following terminology in the discussion while describing the working methodology of PeNCache.

1. *N-hop Neighborhood:* A set of reachable nodes within N number of hops from the given source node. In Figure 4.1, the routers inside the small circle represent the 1-hop neighborhood ($R_1$, $R_2$, $R_7$, $R_9$, and $R_{10}$), while the routers within the larger circle represent the 2-hop neighborhood ($R_3$, $R_4$, $R_5$, $R_8$, and $R_{11}$) of $R_6$.

2. *On-path Routers:* Routers that are located on the request forwarding path. For example, in Figure 4.1, *Consumer1* requests C1, and the *Content Source* serves it. The Interest packet takes the best route to reach the content provider according to the FIB entry. In this case, the request is forwarded via $R_1$, $R_6$, $R_9$, and $R_{11}$, which are the on-path routers for this Interest/Data exchange.

3. *Off-path Routers:* Routers that are located outside the transmission path. For example, in Figure 4.1, $R_5$, $R_9$, and $R_{11}$ are the on-path routers for exchanging the Interest/Data between the Consumer5 and *Content Source*. In this scenario, all other routers outside this set are considered off-path routers for this communication path.

### 4.3.3 Interest Forwarding

As NDN supports multipath forwarding, routers along the path can retrieve content from the neighborhood routers located outside the request forwarding path. While multipath forwarding enhances caching performance, it also introduces additional communication overhead. eNCache [30] introduced a lightweight cooperative mechanism for content searching while forwarding requests towards the content source by modifying the Interest packet structure. Similar to eNCache, PeNCache uses the modified NDN Interest packet structure as shown in Figure 4.1 to enable on-path and off-path request routing. Along with visited nodes (*VisitedNodes*) information, PeNCache also adds the global popularity (*GlobalCount*) of the requested content in the Interest packet before forwarding Interest to the next hop on-path router.

The *VisitedNodes* field of the on-path Interest packet contains information about nodes already explored by the previous $R_{i-1}$ hops, ensuring that the next-hop router $R_i$

**Algorithm 4.1** Interest Packet Processing at a Router $R_i$

1: $RCT \leftarrow \{\}$ /* Request Count Table */
2: $LocalCount(C_i) \leftarrow 0$
3: Interest Packet $I$ arrives at $R_i$ for $C_i$
4: **if** $R_i$ is an On-Path Router **then**
5: **if** $C_i \in$ CacheOf($R_i$) **then**
6: Serve content from $R_i$
7: $LocalCount(C_i)++$ /* Update RCT */
8: **else**
9: $VisitedNodes \leftarrow VisitedNodes \cup$ N-HopNeighborhood($R_i$)
10: $N\_Hop \leftarrow$ N-HopNeighborhood($R_i$) $- VisitedNodes$
11: $I_{OnPath}.VisitedNodes \leftarrow VisitedNodes$
12: $I_{OffPath}.NoPITEntry \leftarrow$ TRUE
13: **for all** Off-Path $R_j \in N\_Hop$ **do**
14: Forward $I_{OffPath}$ to $R_j$
15: **end for**
16: Forward $I_{OnPath}$ to On-Path router $R_t$
17: **if** $C_i \in$ CacheOf($R_k \in N\_Hop$) **then**
18: Serve content from $R_k$
19: $LocalCount(C_i)++$ /* Update RCT */
20: **end if**
21: **end if**
22: Repeat at every On-Path $R_i$ until content $C_i$ is found
23: **end if**

avoids revisiting nodes present in *VisitedNodes* and the *GlobalCount* field passes the global request count of content to the next hop for caching decisions. For example, in Figure 4.1, when $R_9$ receives an Interest packet from $R_6$, it extracts *VisitedNodes* field from the Interest packet, which contains information about routers previously visited by other on-path router(s). In this scenario, $R_2$, $R_7$, and $R_{10}$ have been visited by $R_6$,

and $R_{10}$ is the common neighbor of $R_6$ and $R_9$. Therefore, $R_9$ does not re-explore $R_{10}$. This field reduces communication overhead, especially in dense network topologies where multiple routers share common neighborhoods. During off-path forwarding, the *NoPITEntry* field is set in the Interest packet to indicate that *N-hop* (1-hop, 2-hop, and so on) off-path routers do not add request interface(s) in their standard PIT entries nor forward the request beyond the specified value of $N$. For on-path routers, usual PIT entries for downstream interfaces are created for content caching. Interest packets are simultaneously forwarded to both on-path and off-path neighbors to minimize content retrieval time. Algorithm 4.1 outlines the `PeNCache` Interest forwarding process, while Figure 4.2 illustrates its workflow.

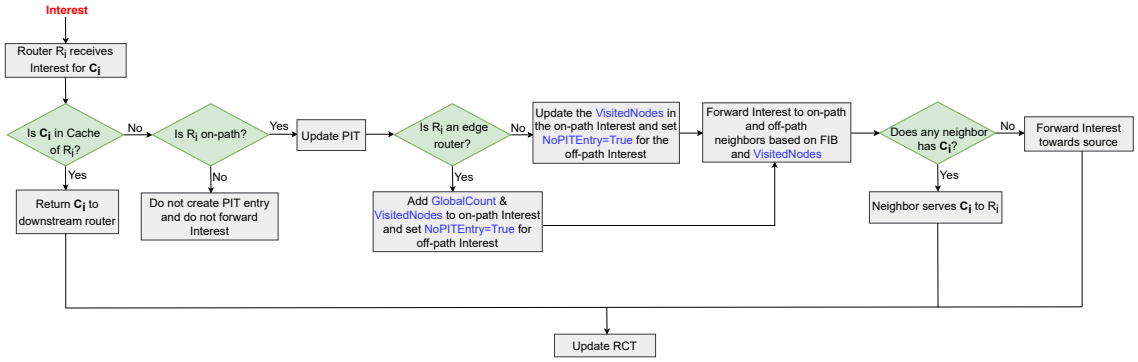



Figure 4.2: Flowchart of Interest Forwarding for Content $C_i$.

### 4.3.4 Popularity Estimation

The popularity estimation of content enhances the utilization of the limited storage capacity of routers by caching the popular content in the network. To improve the cache hit ratio, `PeNCache` considers both the local and global request counts of the content while estimating the popularity, as shown in Equation 4.1.

$$\hat{P}(C_i, R_i) = (1 - \omega) \cdot \text{Local}(C_i, R_i) + \omega \cdot \text{Global}(C_i, R_i) \tag{4.1}$$

where $\hat{P}(C_i, R_i)$ represents the new popularity estimate of content $C_i$ at router $R_i$, $\text{Local}(C_i, R_i)$ represents the local popularity of content $C_i$ at router $R_i$, $\text{Global}(C_i, R_i)$ denotes the global popularity of content $C_i$ at router $R_i$, and $\omega \in [0, 1]$ denotes the weighting factor.

**Algorithm 4.2** Popularity Estimation Among Peers

1: $L \leftarrow \{L_i \mid 1 \leq i \leq n\}$ /* Set of leader nodes */
2: $ER \leftarrow \{ER_i \mid 1 \leq i \leq m\}$ /* Set of edge routers */
3: $RCT \leftarrow \{\}$ /* Request Count Table */
4: $GlobalCount(C_i) \leftarrow 0$
5: **for** each time interval $T$ **do**
6: **for** each edge router $ER_i \in ER$ **do**
7: Exchange $RCT$ with its designated leader node $L_i$
8: **end for**
9: **for** each leader node $L_i \in L$ **do**
10: Exchange $RCT$ with other leaders $\{L_j \mid L_j \in L, j \neq i\}$
11: $GlobalCount(C_i) \leftarrow \text{CountOf}(C_i)_{L_i} + \sum_{j \neq i} \text{CountOf}(C_i)_{L_j}$
12: Update $RCT$
13: Send updated $RCT$ to associated edge routers $ER_i$
14: **end for**
15: **end for**

The content is considered popular for caching when it reaches the dynamic threshold ($\Theta$), which estimates the relative importance of content $C_i$ at router $R_i$ and is determined using Equation 4.2.

$$\Theta = \frac{\max_{i=1}^{M} \hat{P}(C_i)}{\hat{P}(C_i)} \tag{4.2}$$

$\max_{i=1}^{M} \hat{P}(C_i)$ denotes the maximum popularity observed among the $M$ content items requested at $R_i$, $\hat{P}(C_i)$ denotes the popularity of content $C_i$ at $R_i$, and $M$ is the total number of distinct content requests at $R_i$.

*(a) Local Popularity:* Each router individually maintains RCT data structure to track the count of the received Interest packets. The local count of content $C_i$ is incremented in RCT every time the router receives a request for $C_i$, as shown in lines 7 and 19 of Algorithm 4.1. For example, in Figure 4.1, at time $t_1$, $R_1$ receives a request for `C1`. $R_1$ then adds the name of the request to the Interest name column and the corresponding value to the local count column of the RCT.

*(b) Global Popularity:* In the estimation of global popularity, request counts from all consumers in the network are collected, which helps provide a view of the overall demand for various types of content across the entire network. However, global popularity estimation increases the amount of messages exchanged in the network to obtain network-wide information. With the objective of reducing communication overhead, we chose a few routers in the topology called leaders or designated nodes, which are responsible for obtaining popularity data. These leader nodes are selected based on the diameter and connectivity of nodes in the topology using a greedy technique. Following steps are involved in electing a leader node.

1. We first measure the degree centrality of each non-edge router, and then all the non-edge routers are sorted in decreasing order of their degree centrality value.

2. To avoid having leader nodes in close proximity, we compute the minimum distance threshold $d_{\min}$ using Equation 4.3, which ensures that the leader nodes are placed in diverse regions across the network. We mark the first router from the sorted list as a leader node. Then, the next leader node is chosen at a distance $d_{\min}$ from the previously chosen leader node(s), and this process is repeated at every router in the sorted list.

3. After selecting the leader nodes, each edge router is connected to its nearest leader to obtain global popularity information.

$$d_{\min}(L_i, L_j) = \frac{\text{Diameter}(G)}{K} \tag{4.3}$$

In Equation 4.3, $d_{\min}(L_i, L_j)$ indicates the minimum distance between leader nodes $L_i$ and $L_j$. Diameter($G$) denotes the diameter of the topology $G$, and $K$ represents the top $K\%$ non-edge routers with the highest degree centrality, considered as potential candidates for leader nodes in $G$.

Upon receiving a request from a consumer, an edge router adds this local information into the RCT. At every time interval `T`, the edge router forwards the local information to its designated leader to fetch the global request count of the contents. Leader nodes periodically exchange their local information with other leader nodes in

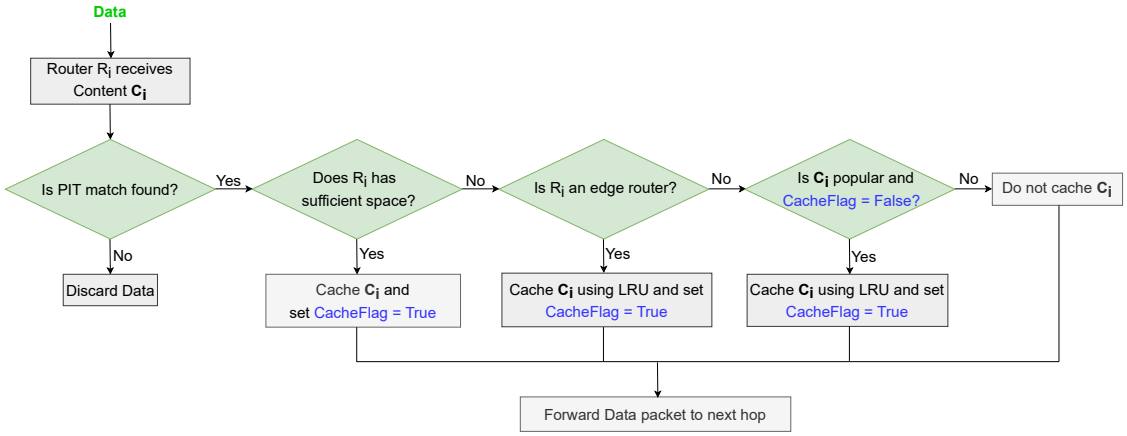



Figure 4.3: Flowchart of Data Packet Forwarding for Content $C_i$.

the network to estimate global popularity. After collecting global data, leader nodes send updated information to their respective edge routers. We restrict the exchange of information about global popularity to periodic intervals of `T` in order to reduce communication overhead. Algorithm 4.2 outlines the process of determining global popularity. Upon receiving the global content count from the leader, the edge router adds the updated count value to the *GlobalCount* field of the on-path Interest packet, as shown in Figure 4.1, before forwarding the packet to the next hop. Routers along the path retrieve global information from the *GlobalCount* field of Interest and estimate content popularity using local and global information, as shown in Equation 4.1. We added *GlobalCount* field in the Interest packet to prevent other routers in the network from coordinating with the leader, which helps reduce communication overhead.

### 4.3.5 Caching Decisions

We adopt the NDN on-path caching approach to perform caching operations quickly with minimal overhead. Algorithm 4.3 presents the caching approach used by on-path routers in `PeNCache`. Each router, upon receiving a Data packet in the downstream direction, makes a caching decision that falls into one of three scenarios:

Firstly, if router $R_i$ is an edge router with regard to the requester, it will always cache the content $C_i$ in order to keep it closer to the requester. Secondly, if router $R_i$

**Algorithm 4.3** Data Packet Processing at a Router $R_i$

```
1: C_i ← New Content at R_i
2: T_{R_i} ← Total capacity of R_i
3: O_{R_i} ← Current occupancy of R_i
4: if C_i ∈ PITOf(R_i) then
5:    if R_i is an Edge Router then
6:       Cache C_i at R_i
7:    else
8:       if T_{R_i} − O_{R_i} ≥ SizeOf(C_i) then
9:          Cache C_i at R_i
10:      else if PopularityOf(C_i) ≥ Θ and C_i ∉ CacheOf(Neighborhood(R_i)) then
11:         Cache C_i at R_i
12:      end if
13:   end if
14:   Forward Data Packet to the next hop without caching
15: else
16:   Discard C_i
17: end if
```

has sufficient space available ($T_{R_i} - O_{R_i} \geq \text{SizeOf}(C_i)$), it caches all passing content $C_i$ to fully utilize the cache capacity. Here, $T_{R_i}$ denotes the total cache capacity of router $R_i$, and $O_{R_i}$ denotes the occupied cache space. Thirdly, if router $R_i$ is a non-edge router and does not have enough space for caching new arrival $C_i$, then $R_i$ caches $C_i$ only if its popularity reaches a dynamic threshold ($\Theta$) and $C_i$ is not available in the cache of any router $R_i$ in the neighborhood. Upon utilizing the full capacity of a router, the `PeNCache` scheme establishes cooperation among neighboring routers to make caching decisions. Cooperation among neighbors helps prevent the caching of the same content in multiple routers, resulting in better cache utilization. In the downstream direction, `PeNCache` does not explicitly coordinate with its neighbors to obtain cache status information. Instead, it employs a lightweight implicit signaling mecha-

nism by introducing the *CacheFlag* field in the Data packet, as shown in Figure 4.1. This option helps to avoid caching the same content at multiple locations along the content delivery path while minimizing the communication overhead. On-path routers utilize the *CacheFlag* field only when their cache capacity is full; otherwise, they update the *CacheFlag* value but do not consider it when making caching decisions. The RCT of the router stores both local and global counts of the content $C_i$ that it receives. The global count value is extracted from the *GlobalCount* field of the Interest packet during the request forwarding to determine the global popularity of $C_i$. The local popularity is measured by the number of times the router has received requests for $C_i$ locally. When the router reaches its capacity, it uses Equation 4.1 to estimate the popularity of the content by extracting information from the RCT. Figure 4.3 illustrates the workflow of the `PeNCache` strategy, highlighting the key steps involved in Data packet forwarding.

For example, consider the network topology shown in Figure 4.1. Assuming the cache capacities of the routers are full, Consumer3 requests content `C1`, which is served by router $R_{10}$. $R_{10}$ adds its ID and sets the cache status to True. This informs the next router in the delivery path, $R_6$, not to cache `C1` because it is available from its direct neighbor. Before forwarding the packet to $R_7$, $R_6$ overwrites the ID and cache status with its own ID and sets the cache status to False. Given that the flag is False, $R_7$ caches `C1` (*if* $PopularityOf(\texttt{C1}) \geq \Theta$) as it is unavailable in its neighbors and forwards it to $R_3$. As $R_3$ is an edge router, it always caches content to minimize access time. By using this flag, a router in the delivery path avoids caching content that is already available in its immediate neighbors when its cache is full. When the cache is full, `PeNCache` employs a replacement policy to create space for new content that meets the caching criteria.

## 4.4 Performance Evaluation

This section outlines the simulation setup, provides details of the network topologies, and discusses the results.

### 4.4.1 Simulation Setup

We performed the simulations using the Icarus caching simulator [135]. In our simulation, $5 \times 10^3$ distinct content items are evenly distributed among multiple sources, and $1 \times 10^5$ requests are collectively generated by all consumers in the network. An initial $10^4$ requests were generated for cache warm-up to ensure the cache reaches a steady state before collecting performance metrics. The popularity of content is modeled using a Zipf distribution [23], with $\alpha$ values ranging from 0.6 to 1.2. Requests from consumers are modeled using a Poisson distribution, with an average arrival of 10 requests/s. Each router can cache the same number of contents, with capacities ranging from 0.1% to 0.5% of the total size of distinct contents in the network, assuming that the size of each content is the same. In the simulation, we use the LRU policy to make room for new content when the cache becomes full. The LRU replacement policy is used because it performs cache lookup and replacement operations quickly without adding much computational overhead [116, 150]. Despite its simplicity, it performs well in terms of caching performance and is widely referred in NDN caching literature [7, 93]. In the simulation, off-path neighborhood is set to *1-hop*. The global popularity exchange between leader nodes is set at one-minute intervals [7, 111] to minimize communication overhead and observe fluctuations in request patterns across the network, and the weight parameter ($\omega$) is set to 0.125 [5, 7]. $K$ in Equation 4.3 is set to 2 to elect the leader nodes. In all the experiments, the link delay between each pair of nodes $(u, v)$ is set to 5 milliseconds. All caching techniques employ Dijkstra's shortest path algorithm to route the Interest and Data packets across the network. We report the results obtained by averaging the data from ten simulation runs, with a 95% confidence interval. The main parameters of the simulation setup are shown in Table 4.1.

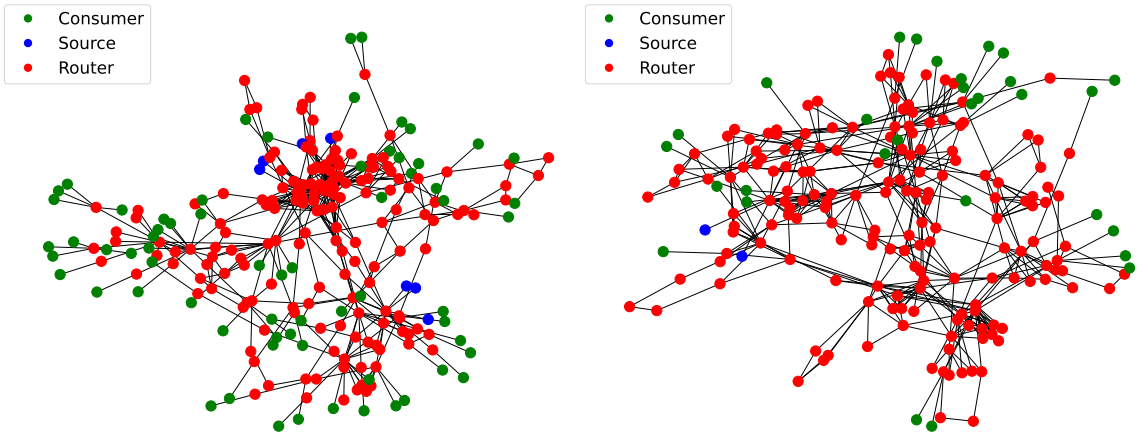
### 4.4.2 Network Topology Setup

We conducted the simulation experiments on two large-scale RocketFuel Internet topologies [146, 147]: Tiscali (AS 3257) and Exodus (AS 3967). Figure 4.4 shows the

Table 4.1: Simulation Parameter Settings

| Parameters | Value |
|---|---|
| Network topology | RocketFuel ISP topologies |
| Content universe (catalog) size | $5 \times 10^3$ objects |
| Number of requests | $10^5$ objects |
| Cache warm-up | $10^4$ objects |
| Content request arrival rate | Poisson distribution, $\lambda = 10$ requests per second |
| Content popularity model | Zipf distribution, $\alpha \in [0.6 - 1.2]$ |
| Router cache capacity | [0.1–0.5]% of content catalog |
| Replacement strategy | LRU |
| Link delay | 5 ms per link |
| Forwarding strategy | Dijkstra's shortest path |
| Experiment repetitions | 5 |

ISP maps of the AS 3257 and AS 3967 topologies, where consumer nodes are shown in green, source nodes in blue, and router nodes in red. These topologies are built in the Icarus simulator using the Fast Network Simulation Setup (FNSS) [133] library.




(a) AS 3257 (b) AS 3967

Figure 4.4: RocketFuel ISP Topologies: AS 3257 and AS 3967.

The AS 3257 topology consists of 240 nodes in total. Out of these, 67 nodes with degree one are designated as consumers, 7 nodes with degree one are designated as

Table 4.2: Details of Network Topologies

| Topology | Nodes | Edges | Consumers | Routers | Sources | Diameter |
|---|---|---|---|---|---|---|
| Tiscali (Europe) AS 3257 | 240 | 404 | 67 | 166 | 7 | 14 |
| Exodus (US) AS 3967 | 201 | 434 | 30 | 169 | 2 | 11 |

content providers (origin content sources), and 166 nodes with a degree greater than one are marked as content routers for caching. In the AS 3967 topology, there are 201 nodes with 30 consumers, 2 origin content sources, and 169 routers. This difference in number of consumers and content sources for these two topologies ensure a more realistic scenario as different number of consumers and providers will have different impact on the caching. All routers in the network have cache enabled, which allows them to store content based on the installed storage capacity. The details of these topologies are shown in Table 4.2.

### 4.4.3 Simulation Results and Observations

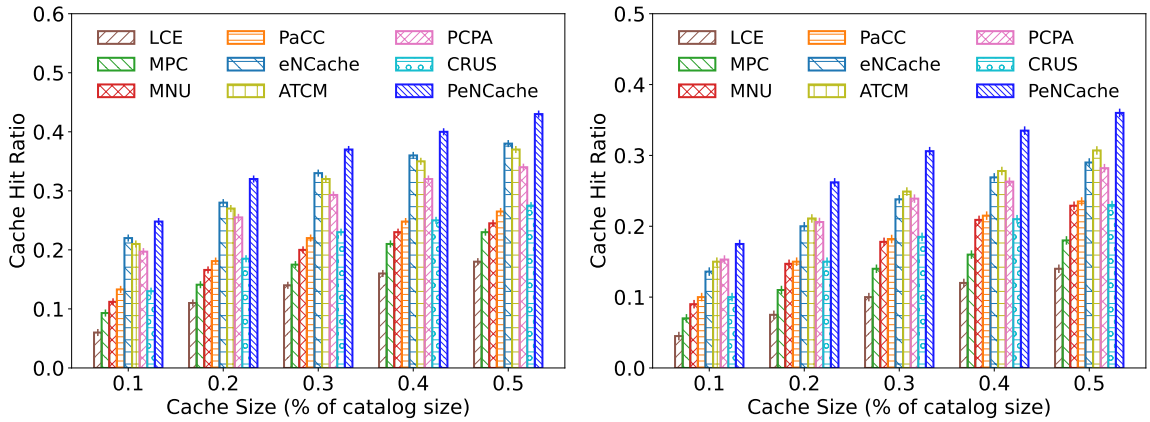
This subsection presents the simulation results and provides a comparative analysis of the nine caching techniques.

We compare `PeNCache` with two conventional caching techniques, LCE [84] and MPC [19], and five advanced techniques from recent studies: MNU [75], PaCC [8], ATCM [124], PCPA [101], and CRUS [93]. In addition, our previously proposed `eNCache` scheme, presented in Chapter 3, is also included for comparison. The details of these caching techniques are discussed in Section 4.2. Similar to eNCache, `PeNCache` is evaluated using four performance metrics: cache hit ratio, content access time, cache diversity, and average hit distance, as mentioned in Chapter 3, where we evaluate the performance by varying cache capacities, number of requests, and request patterns.

***Impact of Varying the Cache Sizes:*** First, we study the impact of cache size on various performance metrics. We vary the cache size between 0.1% and 0.5% of the total distinct content (catalog size), with the popularity parameter $\alpha$ fixed at 0.8 (moderate popularity) [112, 118].

Figures 4.5a and 4.5b illustrate that increasing the cache size of routers results

in higher cache hit ratios for all nine caching techniques. Due to cooperative mechanisms, eNCache, ATCM, and PeNCache outperform other caching techniques. Furthermore, by combining cooperative searching and caching with content popularity awareness, PeNCache achieves up to a 10.35% higher cache hit ratio than eNCache and 18.52% higher than ATCM. It also performs over 80% better than the popularity-based schemes CRUS, PaCC, and 26.47% better than PCPA on the AS 3257 topology. A similar trend is observed on the AS 3967 topology, where PeNCache attains up to 21.35% higher cache hit ratio than eNCache, 24.17% higher than ATCM, up to 70% higher than CRUS and PaCC, and 27.66% higher than PCPA.

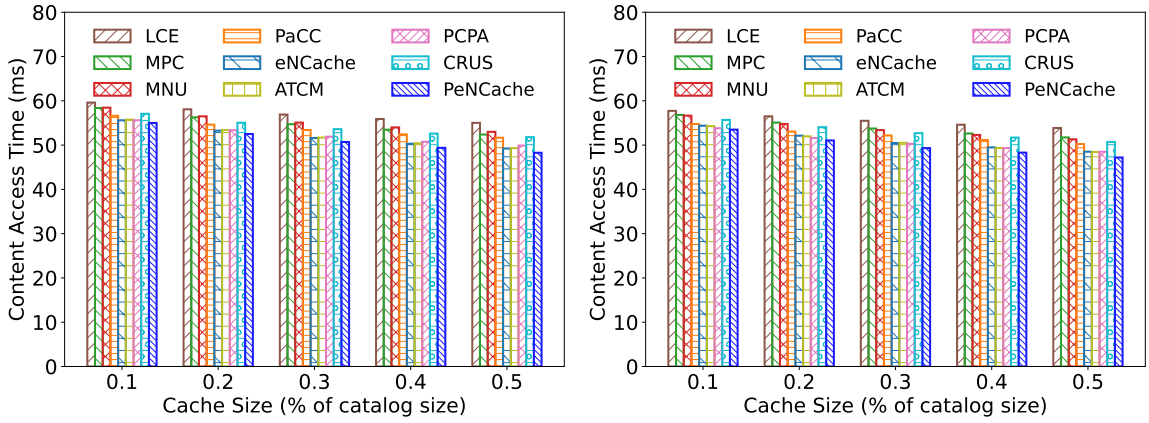



(a) AS 3257 (b) AS 3967

Figure 4.5: Cache Hit Ratio for Different Cache Sizes on Different Network Topologies (Zipf $\alpha = 0.8$).

Figures 4.6a and 4.6b illustrate the content access time of nine caching methods on AS 3257 and AS 3967 topologies. Regardless of the cache size, the content access time of the PeNCache method is significantly lower compared to the other eight approaches on both topologies. The content access time of the PeNCache method is up to 12.3% lower than LCE, 7.85% lower than the popularity-based techniques MPC, 6.8% lower than CRUS, 6.55% lower than PaCC, 3.27% lower than PCPA, and 2.15% lower than the cooperative scheme ATCM on the AS 3257 topology. Similarly, on the AS 3967 topology, PeNCache achieves up to 12.37% lower access time compared to LCE, 8.75% lower than MPC, 6.9% lower than CRUS, 6.07% lower than PaCC, 2.71% lower than

PCPA, and 2.53% lower than ATCM.

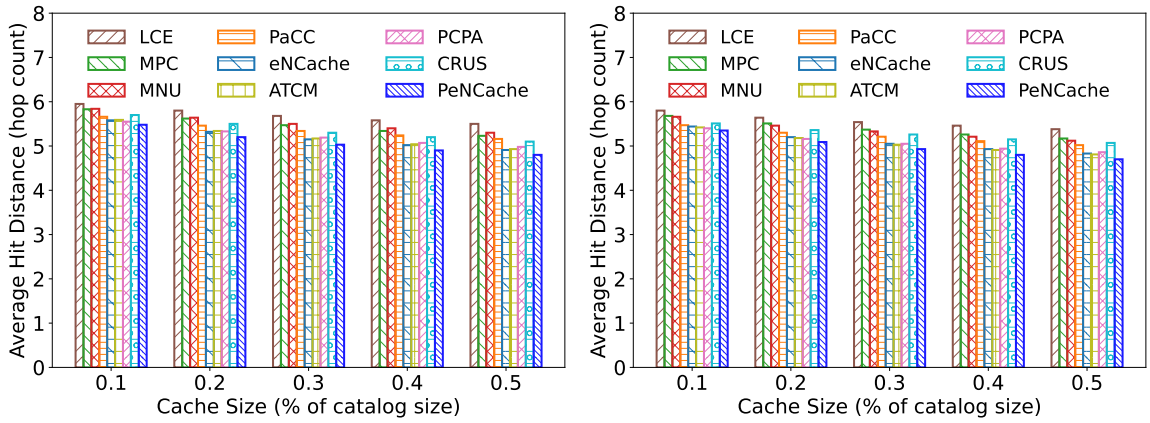



(a) AS 3257 (b) AS 3967

Figure 4.6: Content Access Latency for Different Cache Sizes on Different Network Topologies (Zipf $\alpha = 0.8$).

Figures 4.7a and 4.7b show the average hit distance of all nine caching techniques across both network topologies. As the cache size increases, the hit distance for all caching techniques decreases significantly, as more content is served from the caches of nearby routers. Due to cooperative content searching and caching, the hit distance of `PeNCache` is considerably lower than other cooperative and non-cooperative caching techniques. The consideration of local and global popularity estimation into `PeNCache` enhances the utilization of the router cache by keeping frequently accessed content closer to consumers.

***Impact of Varying the Popularity Parameter*** $\alpha$***:*** To evaluate the performance of caching techniques under different popularity levels, we varied the Zipf $\alpha$. Figures 4.8, 4.9, and 4.10 present the simulation results for all nine caching techniques in terms of cache hit ratio, content access time, and hit distance across both topologies. The simulations vary the value of the popularity parameter $\alpha$ from 0.6 (lower popularity) to 1.2 (higher popularity), with the cache size set to 0.3% of the total distinct content. The parameter $\alpha$ represents the skewness of a Zipf content popularity distribution, where lower values of $\alpha$ indicate a more uniform request distribution, and larger values represent a highly skewed popularity pattern in which a small number of



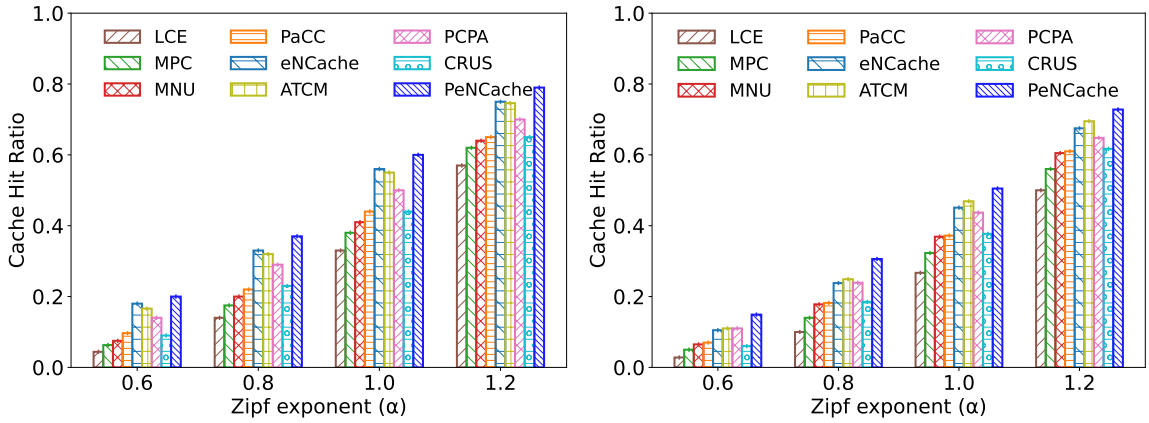

(a) AS 3257 (b) AS 3967

Figure 4.7: Average Hit Distance for Varying Cache Sizes Across Different Network Topologies (Zipf $\alpha = 0.8$).

contents dominate the requests. Values of $\alpha$ in the range $[0.6, 1.2]$ are commonly used in the NDN/ICN literature [107, 112, 118, 141] to model content request patterns. Consequently, increasing the value of $\alpha$ improves the performance of all nine caching techniques on both topologies, as higher $\alpha$ leads to a concentration of requests on a few popular content items that are more likely to be available in router caches.

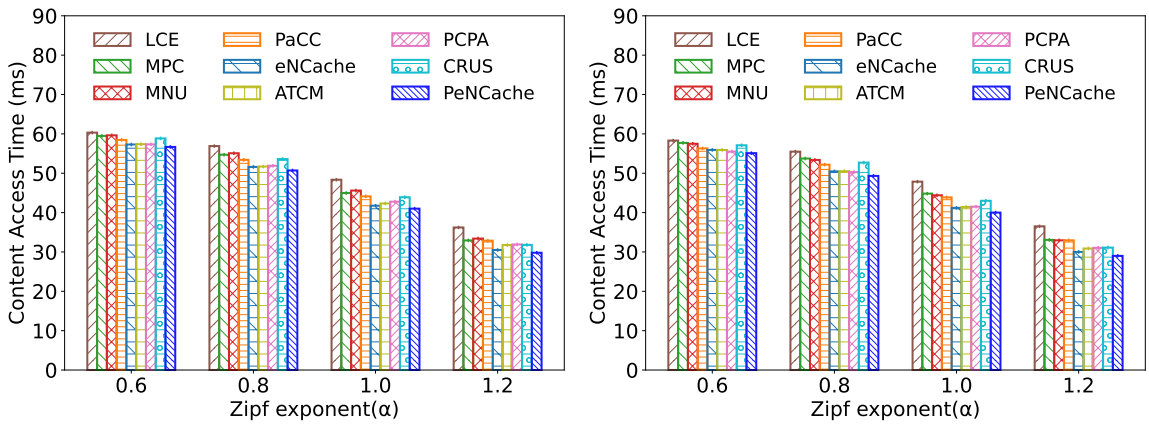
Figures 4.8a and 4.8b compare the cache hit ratio of the nine caching techniques on AS 3257 and AS 3967 topologies, respectively. As shown in Figure 4.8, increasing the popularity parameter $\alpha$ results in a higher cache hit ratio across all nine caching schemes in both topologies. Regardless of the $\alpha$ value and the size of the topology, `PeNCache` consistently outperforms both popularity-based and popularity-agnostic caching techniques.

Figures 4.9a and 4.9b compare the content access time of these caching schemes across both topologies. As the alpha value increases, the content access time for all caching strategies decreases, as most of the content becomes available on nearby routers. Due to its cooperative searching and caching mechanisms, `PeNCache` consistently achieves lower access times compared to both cooperative and non-cooperative approaches. Similarly, Figures 4.10a and 4.10b show that the average hit distance of `PeNCache` is always lower than the other caching techniques, further demonstrating its




(a) AS 3257 (b) AS 3967

Figure 4.8: Impact of Zipf $\alpha$ on Cache Hit Ratio Across Different Network Topologies (Cache Size = 0.3%).

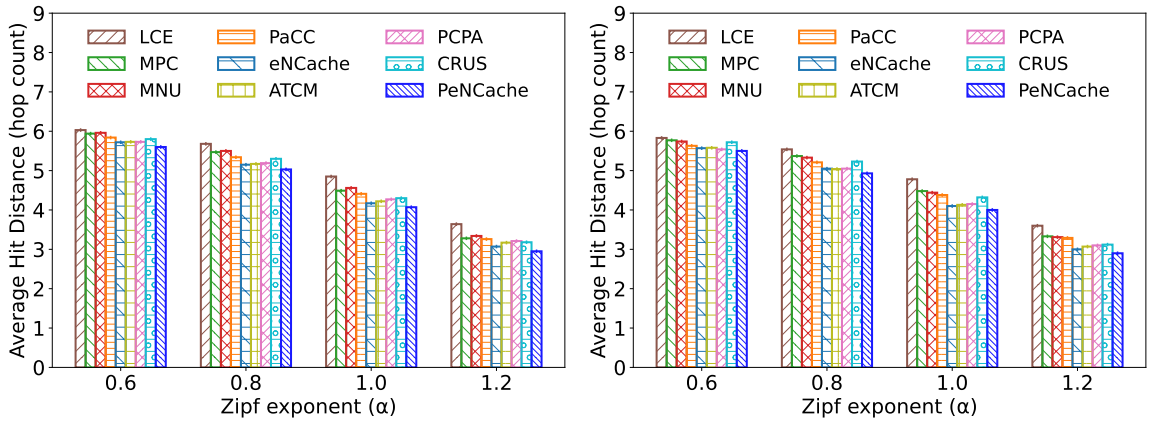



(a) AS 3257 (b) AS 3967

Figure 4.9: Impact of Zipf $\alpha$ on Content Access Time Across Different Network Topologies (Cache Size = 0.3%).

superior performance. It is evident that `PeNCache` consistently outperforms the other eight caching schemes due to its cooperative content searching and caching approach, as well as its consideration of content popularity.

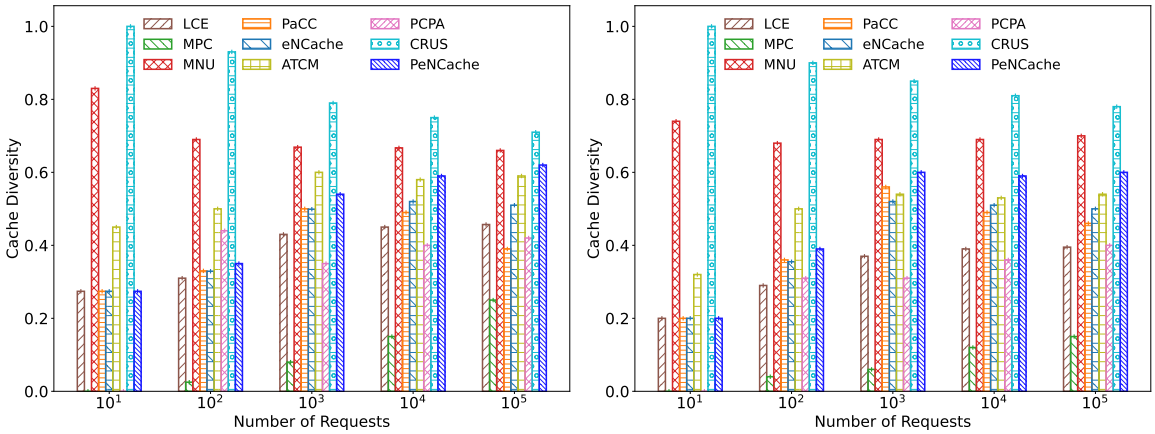
***Impact of Varying the Number of Requests:*** The cache diversity of the network changes over time. Therefore, we computed the cache diversity by varying the number of requests (ranging from $10^1$ to $10^5$). Figures 4.11a and 4.11b show the unique content




(a) AS 3257 (b) AS 3967

Figure 4.10: Impact of Zipf $\alpha$ on Average Hit Distance Across Different Network Topologies (Cache Size = 0.3%).

cached in the network compared with the total cached content for topologies AS 3257 and AS 3967, respectively. From Figure 4.11, we can observe that the cache diversities of all nine caching strategies fluctuate as the number of requests changes.




(a) AS 3257 (b) AS 3967

Figure 4.11: Cache Diversity Comparison Across Different Network Topologies with Varying Requests ($\alpha = 0.8$, Cache Size = 0.1%).

The cache diversity of MPC is lower compared to the other techniques because MPC caches content only when the request count for the content reaches the pre-

defined threshold value. This also results in the underutilization of available router capacity and hence reduces cache diversity. In the beginning, when the number of requests generated by consumers is significantly less (i.e., $10^1$), the cache diversities of `PeNCache`, eNCache, CEE, and PaCC are equal since all methods cache all the passing content in all the routers along the content delivery path when sufficient cache storage is available. However, as the number of requests increases, `PeNCache` outperforms eNCache, CEE, PaCC, PCPA, and ATCM. In terms of cache diversity, CRUS and MNU perform better than the other caching techniques, as they cache content in only a few routers along the content delivery path, resulting in a higher ratio of unique content to total cached content. However, CRUS and MNU may underutilize the available cache space, resulting in a lower cache hit ratio. It is worth noting that improving cache diversity does not necessarily lead to better performance in terms of cache hit ratio and content access time. Therefore, caching decisions should be made carefully by considering the availability of cache resources.

For example, `PeNCache` caches content `C1` and `C2` in all routers $R_1$, $R_6$, $R_9$, and $R_{11}$ while being delivered back to the requesting consumer to better utilize the available storage space, resulting in four copies of each `C1` and `C2` available in the network, and since the total number of unique contents is two, the cache diversity ratio is 0.25. This means that 25% of the total cached content is unique. Whereas the MNU strategy caches content in a few selected routers based on the utility value. For example, content `C1` is cached at $R_6$ and $R_9$, and `C2` is cached at $R_6$ while being delivered back to the requesting consumer. In this scenario, there are two copies of `C1` and one copy of `C2` available in the network. The total number of unique contents remains two, resulting in a cache diversity of approximately 0.67. This means that approximately 67% of the total cached content is unique. Similarly, based on probability, CRUS caches content in a few routers along the delivery path, resulting in higher diversity.

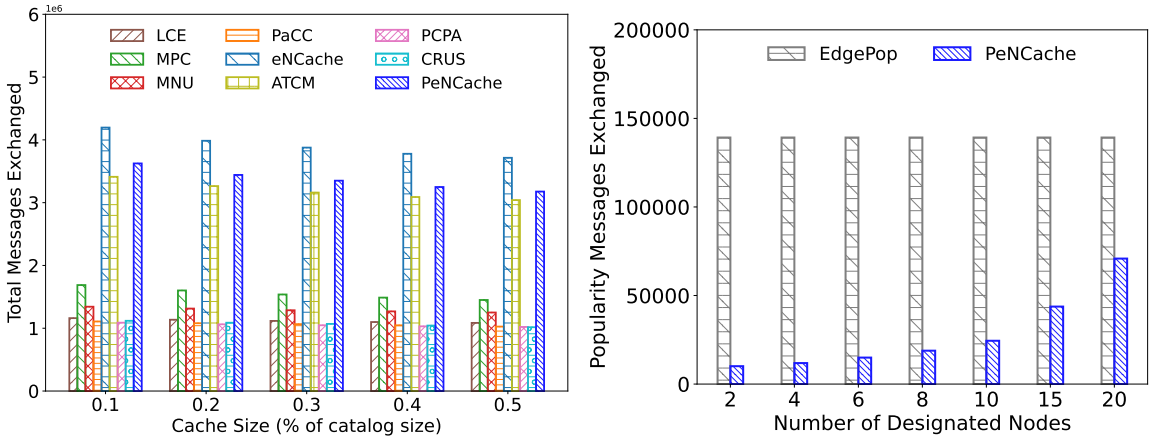
### 4.4.4 Overhead Analysis

The cooperative mechanism used by `PeNCache` to explore the availability of content in the neighborhood improved the cache hit ratio and reduced the latency. As discussed

in Chapter 3 with eNCache, cooperative methods introduce additional communication overhead resulting from the exchange of Interest/Data packets during cooperation. Similarly, PeNCache also introduces overhead due to cooperation. This overhead consists of the number of additional messages sent to *N-hop* off-path router(s) $R_j$ when a cache miss occurs at an on-path router $R_i$. PeNCache also accounts for content popularity, where edge nodes are required to exchange information with designated nodes to determine the global popularity of the content, introducing additional message exchanges in the network. To analyze the communication overhead of PeNCache, we conducted simulation studies on the AS 3967 topology using the same setup and parameters as listed in Table 4.1.

Figure 4.12a shows the comparison of the total number of messages exchanged in the network to satisfy all consumer requests. The total number of messages indicates the sum of messages sent and received within the network during the simulation. These messages include Interest, Data, and any additional messages sent for routing, caching, replacement, or popularity updates. We measured the number of messages exchanged by varying the cache size with a fixed $\alpha$ value of 0.8. From Figure 4.12a, it can be seen that the conventional on-path request routing approaches CEE, MPC, MNU, PaCC, PCPA, and CRUS have less number of messages being sent compared to cooperative techniques eNCache, ATCM, and PeNCache. This is because CEE, MPC, MNU, PaCC, PCPA, and CRUS only forward the Interest to the next-hop on-path router based on the FIB entry to obtain the content. However, MPC and MNU result in larger message exchanges compared to the other on-path approaches (CEE, PaCC, PCPA, and CRUS). Although MPC and MNU are on-path techniques, they still require exchanging a few messages in the network to make their caching decisions. In MNU, the content producer sends the request to the network manager in order to get the best potential router(s) along the content delivery path for caching. The network manager assesses the topological and dynamic characteristics of the routers and responds to the content producer with the set of routers responsible for caching. In MPC, additional messages are sent by the on-path router to all its neighbors upon caching popular content, which leads to higher message transmission

in highly connected topologies. In the ATCM strategy, cache information exchange results in higher communication overhead because whenever a router caches or replaces content, it sends information to its neighbors. The overhead of ATCM and PeNCache are comparable, though PeNCache incurs slightly higher overhead due to the exchange of popularity information. However, due to the integration of content popularity in the caching decision, PeNCache outperforms eNCache. Because PeNCache caches globally popular content in the network, it reduces the number of Interest messages exchanged to retrieve the content.




(a) Total Messages Exchanged (b) Messages for Popularity

Figure 4.12: Communication Overhead of Different Caching Techniques on the AS 3967 Topology.

PeNCache considers both the local and global popularity of content while making caching decisions. Each router individually tracks content requests to estimate local popularity, whereas a few designated nodes in the network coordinate with nearby edge routers to estimate global popularity. Exchanging information between the edge nodes and designated nodes, along with the coordination among the designated nodes to obtain global information, adds additional communication burden to PeNCache compared to the local popularity estimation strategies MPC, PaCC, PCPA, and CRUS.

To assess the amount of information exchanged between nodes to get global information, we compare the PeNCache with another global popularity estimation technique

proposed in the literature [50]. We refer to the global popularity estimation approach proposed in [50] as EdgePop. The EdgePop method establishes communication among edge nodes to obtain global information. At the end of each cycle, each edge node exchanges its local information with the other edge nodes in the topology. In Figure 4.12b, for popularity estimation overhead, we omitted the CEE, MNU, MPC, PaCC, PCPA, ATCM, eNCache, and CRUS strategies since they either did not consider content popularity or did not send additional signals for popularity estimation. CEE, MNU, and eNCache strategies do not maintain any popularity table, whereas MPC, PaCC, PCPA, ATCM, and CRUS strategies locally measure the request count without sharing information with other routers.

Figure 4.12b shows the comparison of the overhead associated with global popularity estimation for `PeNCache` and EdgePop. The X-axis represents the number of designated nodes for `PeNCache` strategy, and the Y-axis represents the number of messages exchanged to obtain global popularity. For both approaches, message exchanges among nodes occur at intervals of time `T`. From Figure 4.12b, it can be seen that `PeNCache` exchanges fewer messages than EdgePop (an edge-based global popularity estimation approach). As the number of designated nodes increases for the `PeNCache`, the number of packet exchanges slightly increases. However, even with a larger number of designated nodes, `PeNCache` outperforms EdgePop. The reason behind the lower communication overhead of the `PeNCache` scheme is that edge nodes only need to coordinate with the nearest designated node. Whereas for EdgePop, each edge node needs to communicate information to all other edge nodes, which increases communication overhead. For example, if there are $n$ edge routers in the topology, then the total number of messages exchanged is $n \times (n-1)$. Moreover, exchanging information among edge nodes becomes challenging in larger network topologies. This is because one edge node is usually far from the others, making it difficult to exchange popularity information.

*Overhead Analysis of `PeNCache` with Varying Designated Nodes and Information Exchange Frequency:* Figure 4.13 shows the impact of the varying number of designated nodes and the frequency of information exchange on the communication over-

head. From the figure, it can be observed that as the number of designated nodes increases, the number of messages exchanged in the network also increases. This is because each designated node requires information exchange among them. However, a higher number of designated nodes helps balance the load on each node for popularity estimation and also decreases the distance between edge routers and designated nodes, as well as among the designated nodes. For example, if there are two designated nodes in the network, the messages exchanged occur only between these two, which reduces communication overhead. However, this configuration can increase the burden on those two nodes and also extend the time to exchange information if they are located far apart in the network. Additionally, more frequent information exchanges can lead to higher communication overhead as nodes frequently share their popularity information with others. Therefore, it is important to maintain a balance between information exchange and the selection of designated nodes in the network, depending on the size of the topology and consumer demand.

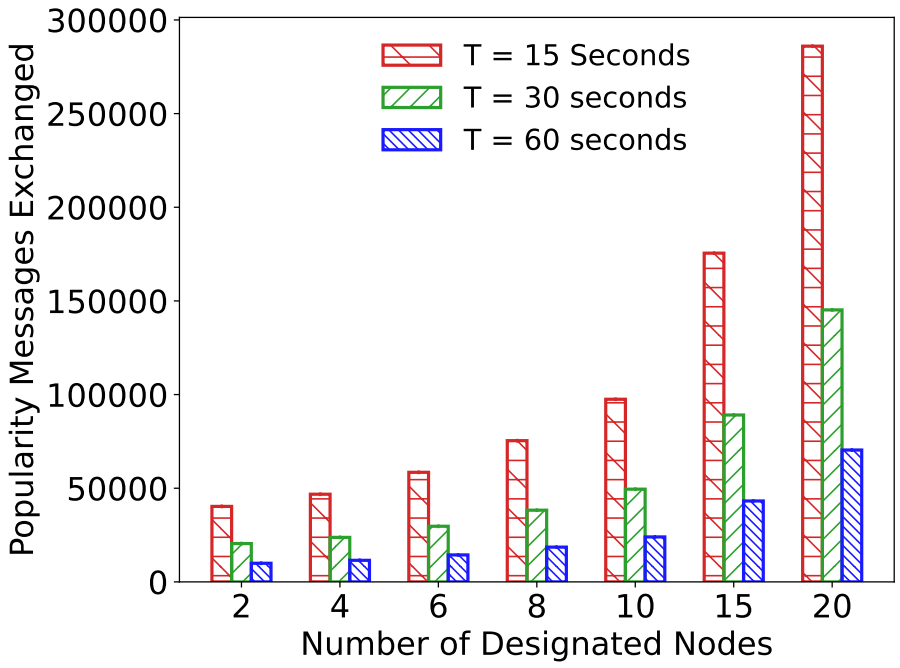



Figure 4.13: Overhead of PeNCache with Varied Number of Designated Nodes and Exchange Frequency on the AS 3967 Topology.

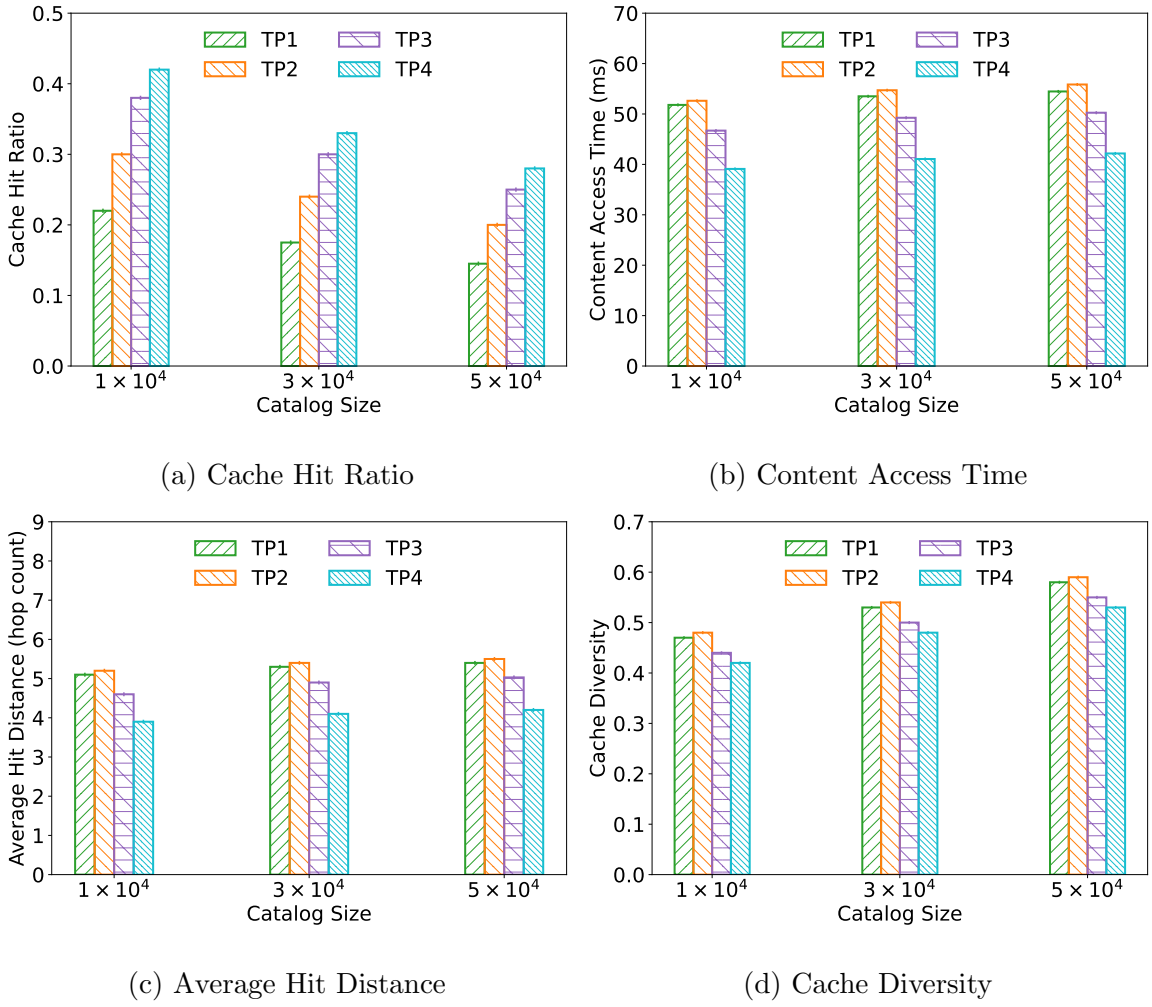
### 4.4.5 Scalability Analysis of PeNCache

While the previous results showed that PeNCache performs well compared to other methods. Another important practical consideration is the scalability of the method. To assess the scalability of PeNCache, we conducted simulations with various network sizes and also by varying catalog sizes. We performed simulations with 4 ISP topologies of different sizes. These topologies were taken (parsed with the ISP map files) from the RocketFuel datasets [146, 147] and integrated into the Icarus simulator for performance analysis. In each topology, consumers and sources are one-degree nodes/routers that are connected to routers with a degree greater than one. Routers with a degree greater than one are considered NDN routers responsible for content caching. This is because an edge router has at least two connections, one to a consumer and the other to a router. The characteristics of these network topologies are summarized in Table 4.3. For simplicity, these topologies are referred to as TP1, TP2, TP3, and TP4 in the graphs. We compared the performance of PeNCache across these four different-sized topologies using the same four performance metrics as in the earlier experiments: cache hit ratio, content access time, average hit distance, and cache diversity. This analysis evaluates how PeNCache performs in large network topologies. The Zipf popularity parameter $\alpha$ is set to 0.8, and the catalog size is configured as $1 \times 10^4$, $3 \times 10^4$, and $5 \times 10^4$. In each topology, each router has a maximum capacity to hold 15 contents. Consumers collectively generate a total of $5 \times 10^5$ requests during the simulation for different contents from the catalog.

Table 4.3: Summary of Characteristics of Rocketfuel ISP Topologies

| Topology | Nodes | Edges | Consumers | Routers | Sources |
|---|---|---|---|---|---|
| TP1: Exodus (US) AS 3967 | 201 | 434 | 30 | 169 | 2 |
| TP2: Tiscali (Europe) AS 3257 | 240 | 404 | 67 | 166 | 7 |
| TP3: Verio (US) AS 2914 | 960 | 2821 | 235 | 711 | 14 |
| TP4: Level 3 (US) AS 3356 | 3447 | 9390 | 2280 | 1082 | 85 |

Figures 4.14a and 4.14b show the cache hit ratio and content access time of PeNCache, respectively. Our findings show that as network size increases, PeNCache




(a) Cache Hit Ratio

(b) Content Access Time

(c) Average Hit Distance

(d) Cache Diversity

Figure 4.14: Performance of PeNCache on four different-sized network topologies with varying catalog sizes.

performs better in terms of cache hit and content access time. In larger topologies, the availability of more routers for content caching results in a higher number of cached content items across the network. Larger network topologies are highly connected, benefiting the cooperative searching of PeNCache and enabling more efficient content retrieval. This is because there is a higher chance that the requested content will be available on nearby off-path routers. Furthermore, smaller catalog sizes result in better performance in terms of cache hit ratio and latency, while larger catalog sizes lead to decreased performance. This is because router cache memory is limited, so with a smaller catalog size, PeNCache is able to cache the most popular content in the

network to satisfy the majority of consumer requests. However, with larger catalog sizes, consumers request more diverse content, causing the cache capacity of routers to saturate quickly. As a result, routers frequently replace cached content to accommodate new arrivals. This leads to higher cache misses and latency, as most content is unavailable in the cache and needs to be fetched from the original source. A similar observation is seen for the average hit distance, as shown in Figure 4.14c.

Figure 4.14d shows the cache diversity across all four topologies with varying catalog sizes. The cache diversity increases as the catalog size increases (across the topologies). This is because consumers are more likely to request a wider variety of content, resulting in different routers caching more diverse content. Furthermore, Figure 4.14d shows that topologies with fewer content routers (TP1 and TP2) achieve better cache diversity than those with more routers (TP3 and TP4). This phenomenon is due to the over-caching of certain items in the network, which increases the cache hit ratio because routers can accommodate a larger number of contents. However, this comes at the cost of reduced content diversity in the network. In larger topologies, multiple routers may cache the same popular content due to high demand. This leads to a situation where the total number of cached contents exceeds the proportion of unique content available in the network. For example, consider a network topology, say Topo1, with 10 routers, where each router can hold 10 contents. This network can cache a maximum of 100 contents. If 100 contents are cached network-wide, and 20 of them are unique, then the cache diversity of this topology is 20%. Let's consider another topology, say Topo2, which has 5 routers, each capable of holding 10 contents. The maximum number of contents this network can cache is 50. If 50 contents are cached network-wide and 20 of those are unique, then the cache diversity of Topo2 is 40%. This demonstrates that caching the same popular content at multiple locations in topologies with a higher number of routers increases redundancy, which improves the cache hit ratio and reduces latency but lowers cache diversity.

`PeNCache` offers notable improvements in cache hit ratio and latency when scaling to larger network topologies and increasing traffic. However, it is important to note that as the network size grows, each router may connect to several other routers. This

may result in an increased communication overhead for PeNCache due to its cooperative searching and popularity estimation mechanisms. To minimize communication overhead, PeNCache modifies the NDN Interest and Data packets. These modifications help reduce message transmission in the network. Another challenge in complex network topologies is managing the popularity table and computing the popularity of each content item. To reduce this overhead, PeNCache exchanges popularity information at periodic intervals.

## 4.5 Discussion

Several practical considerations drive the deployment decisions of PeNCache. Some of the important considerations are as follows.

*1. Network Topology*: It is worth noting that the performance of PeNCache is influenced by the structure of the network topology, the number of edge routers, and the number of selected leader nodes. If the underlying topology is dense with many edge routers, then the overhead incurred for exchanging the popularity tables will be higher. However, this overhead is not unique to PeNCache; while all other caching techniques using popularity will also have similar overhead. Further, PeNCache exhibits reduced overhead compared to many other methods as discussed in 4.4.4.

*2. Handling Failures*: It is not uncommon for nodes to fail in the network. Such failures can be of designated nodes or other nodes. These two types of failures need to be handled differently. If a leader node fails, all edge routers associated with it must find an alternative leader node for reassociation. It is worth noting that such information is already available to the edge routers, as they select a leader node that is in close proximity for association. In this case the edge routers can request the next nearest leader node for association. On the other hand, if a non-leader node fails, the default NDN fault tolerance mechanism can be used [178]. In such cases, the network can continue to operate normally because NDN replicates content across multiple routers (in-network caching) and supports multipath forwarding, allowing content to be retrieved by rerouting requests through alternative paths. This ensures

minimal disruption in the network. By default, NDN nodes use timeout mechanisms and retransmissions to detect failures. If an Interest times out and the corresponding Data packet is not received within that time, routers can re-express the Interest over alternate paths. It is also worth noting that, even for load balancing purposes, nodes may maintain multiple entries as next-hop nodes to reach a destination.

*3. Varying Content Sizes*: In the current evaluation of `PeNCache`, all content chunks are assumed to be of the same size, although this assumption does not always hold in practice [18, 25, 142]. Content of varying sizes can be handled by `PeNCache` with minor modifications to the content replacement algorithm, where more than one existing chunk may need to be evicted to accommodate a newly arriving content. Handling variable-sized content in caching systems has been studied in prior work [18, 25, 82, 142], where replacement decisions are adjusted to evict either multiple smaller chunks or a single chunk of sufficient size to free the required cache space when accommodating a larger incoming content. For example, if a content `Name3` of size $X$ bytes arrives at router $R_i$, the router may evict contents `Name1` and `Name2` to free at least $X$ bytes of cache space. Such changes can be applied to the replacement logic of `PeNCache` without affecting its overall design.

*4. Content Updates*: NDN packets have a freshness parameter embedded in them. This is used to assess whether the content is useful or otherwise. If updated content is available, then it will eventually percolate into the network. It is up to the application to decide whether a data chunk can be used. If there are very frequent updates to the data chunks of all sources (which is unlikely) then the performance will be affected for all caching methods including `PeNCache`. In fact, for other caching techniques, the impact will be more as they do not use cooperation like `PeNCache`. Cooperation minimizes the number of places where cached content is invalidated compared to other methods.

Let us consider an example to understand how frequent content updates can affect the performance of caching techniques. In Figure 4.15, Consumer1 requests content `C1`, and the request travels through the path *Consumer1,* $R_7, R_4, R_2, R_1$, *Source*. The source responds with the requested content `C1`. In a non-cooperative caching approach,

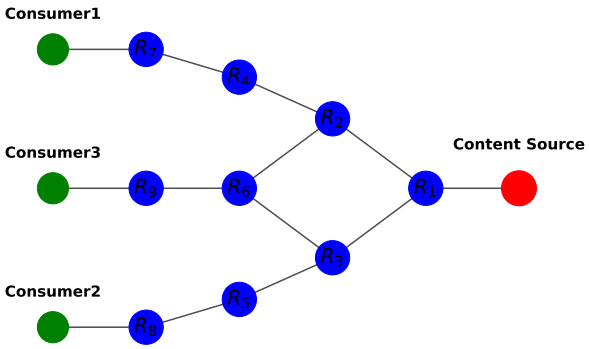



Figure 4.15: Example Network Topology.

all routers along this path independently cache the received content. Now, suppose the source updates content `C1`. Without cooperation, all routers that have cached the previous version must invalidate their copies, resulting in invalidation operations at four different routers in this example. On the other hand, `PeNCache` enables routers to cooperate while making caching decisions. Such cooperation reduces the total number of cached copies in the network and, consequently, minimizes the number of invalidations required when content updates occur. For example, if only routers $R_1$ and $R_7$ store `C1`, then invalidation would be required at only these two routers.

*5. Interest Packet Overheads:* Like `eNCache`, `PeNCache` uses the *VisitedNodes* field to prevent revisiting nodes explored by previous-hop router(s). As a result, `PeNCache` also incurs an overhead due to the *VisitedNodes* field in the Interest packet, which can grow with each hop depending on the number of neighbors.

*6. Packet Loss and Duplicate Interest Packets*: When Interest packets are sent to neighboring nodes, either on-path or off-path, there is a chance they might not reach all intended recipients due to potential packet loss. Similarly, Data packets transmitted by neighboring nodes may also fail to arrive at the destination. However, this situation is not a major concern, as it is a common issue in networks and can be managed effectively. Packet losses along the on-path route are addressed through NDN default fault handling mechanisms using NACK or timeout. In the case of off-path losses (Interest or Data), it is assumed that the neighbor does not have the requested content, and the content is then retrieved from other on-path routers or from the original

content publisher. For example, let's consider a scenario where an off-path packet is lost. In Figure 4.1, Consumer1 requests the content `C1`, which is provided by the *Content Source*. The Interest packet follows the best route strategy, moving toward the provider via $R_1$, $R_6$, $R_9$, and $R_{11}$. Assume that `C1` is available in the caches of $R_2$ and $R_{11}$. When the packet arrives at the on-path router $R_6$ from $R_1$, it simultaneously forwards packets to its next hop on-path ($R_9$) neighbor and all 1-hop off-path neighbors ($R_2$, $R_{10}$, and $R_7$). If the packet from $R_6$ to $R_2$ is lost due to link failures or transmission errors, $R_6$ does not wait for a response from $R_2$. Instead, it continues to forward the Interest packet toward the source and successfully retrieves `C1` from $R_{11}$, which is then returned to the downstream router $R_1$. Furthermore, it is possible for a router to receive Interest packets for the same content from multiple neighbors. However, this does not cause any problems in `PeNCache` because requests are forwarded simultaneously to all neighbors. The neighbor that first responds with a Data packet to the Interest is forwarded by the router toward the consumer, and the entry is removed from the PIT. Hence, when the router receives the Data packet for the same request later, it will drop that packet.

## 4.6 Summary

In this chapter, we described `PeNCache`, a popularity-based cooperative content searching and caching technique. We modified the NDN packet structure to achieve cooperation for popularity estimation, content searching, and caching. `PeNCache` considers both the local and global popularity of content. For global popularity estimation, it uses a lightweight cooperation mechanism by electing a few designated nodes based on network diameter, degree centrality, and distance. We evaluated the performance of `PeNCache` using a discrete event simulator on large-scale Internet topologies. The simulation results demonstrated that `PeNCache` outperforms other caching techniques in terms of cache hit ratio and content retrieval time. These results suggest that cooperative content searching combined with popularity-aware caching decisions better utilizes the router's cache storage. However, in `PeNCache`, all routers are involved

in content searching and caching, which places a burden on every router. Furthermore, in PeNCache, multiple routers maintain the popularity table for measuring local and global popularity, which adds storage overhead at each router. To optimize content searching, caching, and popularity estimation, in the next chapter we present a caching technique in which these responsibilities are assigned to a few selected nodes in the network.

# Chapter 5

# DPCCache: Demand Driven Content Popularity based Cooperative Caching for NDN

## 5.1 Introduction

As discussed previously, the in-network caching capability of NDN offers improved response times. However, the cache is a premium resource in the network routers and is limited by capacity constraints. Hence, it is important to make better use of the available router capacity to improve the performance of the NDN network. Our earlier works show that cooperative caching techniques offer better cache utilization, but the communication required among network nodes to exchange content/popularity-related information introduces overhead. In this chapter, with the aim of reducing communication overhead and maximizing content caching in the network, we present a community-centric cooperative caching technique named DPCCache. Our approach is to divide the network into smaller communities using a community detection algorithm and elect one leader node within each community. This leader is responsible for managing routing, caching, replacement, and popularity estimation tasks. The community division approach in DPCCache reduces the load on individual routers and

simplifies cooperation. DPCCache considers both local and global patterns of content requests to determine whether the content is popular. Global popularity estimation becomes easier in DPCCache with the help of leader nodes, where each leader exchanges its local popularity information with other leader nodes.

The key contributions of this chapter are as follows:

1. We propose DPCCache, a semi-decentralized approach for collaborative content search, caching, and popularity estimation.

2. DPCCache brings cooperation among the routers in the neighborhood by partitioning the network into several disjoint communities/clusters and elects a node within a cluster as representative, which makes caching decisions for all the nodes within the community.

3. Popularity estimation in DPCCache is demand-driven and is performed only for the required content. Unlike traditional methods that require periodic exchanges among all nodes, DPCCache restricts updates to a single exchange per community/cluster on a demand basis.

4. DPCCache reduces content redundancy by caching unique content within communities. Furthermore, it uses a weighted average for local and global popularity to reduce the chances of caching the same content across all communities.

5. We use the Icarus simulator for the implementation of DPCCache. We compare the performance of DPCCache with state-of-the-art caching techniques over two different realistic network topologies in terms of valuable metrics like cache hit ratio, cache diversity, content access time, and server workload.

The rest of the chapter is organized as follows: Section 5.2 presents an overview of some prominent existing works from the NDN caching literature. Section 5.3 outlines the design of DPCCache, describing the community/cluster formation algorithm, content request handling, content caching, replacement, and popularity estimation techniques. Section 5.4 provides the simulation setup and results. Section 5.5 discusses

the handling of content of varying sizes and presents the corresponding simulation results. Section 5.6 offers a discussion on the strengths and challenges of DPCCache. Finally, Section 5.7 summarizes the chapter.

## 5.2 Prior Work

As we discussed previously, content searching and caching decisions can be made either cooperatively or independently at each router. A more comprehensive analysis of these caching techniques is provided in Chapter 2.

Popularity-Topology Cache (PT-Cache) [77] is a non-cooperative caching technique in which each router independently makes caching decisions. PT-Cache combines content popularity with the closeness centrality of routers to place frequently requested content closer to consumers. However, without cooperation, multiple routers in the network end up caching the same content, which increases content redundancy and leads to poor utilization of cache space. The authors of [124] proposed a cooperative popularity-aware caching technique in which cooperation is established among *1-hop* neighbors. In this approach, each router maintains a table that records content availability within its *1-hop* neighborhood. However, whenever cache updates occur (such as newly cached or discarded chunks), these tables must be exchanged among *1-hop* neighbors to keep information about the available content in the neighborhood up to date, which introduces significant communication overhead. In [173], the authors proposed a cooperative caching technique that creates fixed, predetermined-size clusters (e.g., $3 \times 3$, $4 \times 4$, $5 \times 5$ routers) and assigns content chunks to specific routers within each cluster using a hash function. To reduce flooding overhead, caching updates are advertised only within the cluster. However, this approach is applicable only to equal-sized clusters, and its performance is heavily dependent on the chosen cluster size. In addition, the flooding required for routing or cache updates, even when restricted to clusters, can still introduce significant overhead, especially as cluster sizes grow or in highly dynamic network environments, since each router must flood this information to other routers within its respective cluster.

## 5.3 DPCCache: Design and Working Methodology

In this section, we present the working of the proposed popularity-aware cooperative content caching technique. It is a hybrid approach that combines both local and global popularity estimation to make better use of the limited cache capacity available in the network. In the following subsections, we describe the design and relevant algorithms on neighborhood router identification, content processing (routing), content caching, replacement, and popularity estimation (local and global) decisions made at each router using the reference topology shown in Figure 5.1. The figure also shows that, along with the CS, PIT, and FIB, each leader node in a community maintains an additional data structure to estimate content popularity. This popularity table maintains the frequency of each content item within its respective community.

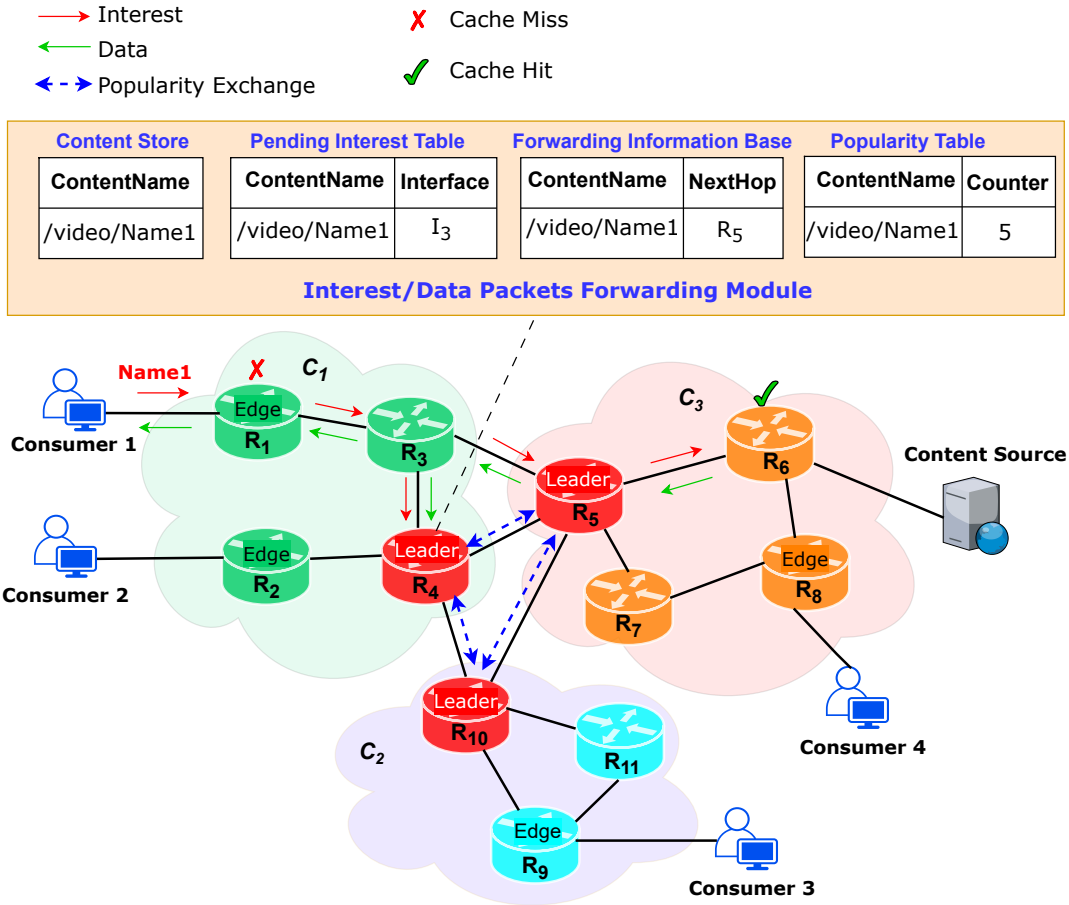



Figure 5.1: DPCCache Reference Architecture.

### 5.3.1 Design Rationale

A key challenge in cooperative content searching and caching in NDN is the communication overhead incurred during information exchange. While cooperation among routers enables better caching decisions and improves cache diversity across the network, establishing and maintaining such cooperation becomes increasingly difficult in large network topologies. Consideration of popularity in caching further enhances performance by keeping frequently requested content in router caches. However, popularity estimation requires measuring the access frequency of content items. Estimating global popularity, which reflects how popular a content item is across all consumers, requires information exchange among routers [50, 186]. As the network size increases, this exchange becomes challenging and incurs substantial communication and storage overhead.

In our previous approach, `PeNCache`, each router maintains a popularity table to track passing content and periodically shares this information with a small set of designated routers to estimate global popularity. Although this approach improves caching efficiency, it relies on relatively ad hoc cooperation among routers. As a result, every router incurs additional storage overhead, and frequent popularity table exchanges among routers introduce significant communication overhead, particularly in large network topologies where routers are often far apart. To address these limitations, `DPCCache` adopts a semi decentralized caching approach to enable structured cooperation through community based organization of the network. The network topology is first partitioned into smaller, non-overlapping communities. Within each community, a router with high degree centrality is elected as a leader node. These leader nodes are responsible for content searching, caching, and popularity estimation. Routers within the same community cooperate only with their designated leader node. This design confines popularity estimation and information sharing to the community level and limits global popularity exchange to once per community rather than among all routers. Unlike the ad hoc cooperation mechanism in `PeNCache`, the structured community formation and leader based coordination in `DPCCache` eliminate redundant

content caching within a community and minimize communication overhead.

### 5.3.2 Overview of DPCCache

Before delving into the design details of DPCCache, we first provide an overview of its working with the help of Figure 5.1.

To begin with, DPCCache partitions the network topology into multiple communities using a community detection algorithm where every node is part of a single community. A leader is then selected within each community based on node connectivity, and the nodes within the community coordinate with their leader for content retrieval and caching decisions. As shown in Figure 5.1, the topology is divided into three communities: $C_1$, $C_2$, and $C_3$, with leader nodes $R_4$, $R_{10}$, and $R_5$ selected for communities $C_1$, $C_2$, and $C_3$ respectively. In DPCCache, request forwarding and caching decisions of every node are managed by the leader node of the community, helping to reduce repeated queries and avoid content redundancy. The content is first searched within the community with the help of the leader. If the content is unavailable within the community, it is retrieved from another community or the original source. For example, in Figure 5.1, when router $R_1$ receives a request for content Name1 from Consumer 1 and a cache miss occurs at $R_1$, the request is forwarded to the leader node $R_4$, which then determines whether any router within community $C_1$ holds the content. Similarly, for caching decisions within $C_1$, when $R_3$ makes a caching decision, it consults leader $R_4$ to avoid content redundancy. As DPCCache ensures no redundancy within a community, it may relocate evicted content from one router to another within the same community to optimize the use of underutilized router space. DPCCache uses a hashing technique for this content relocation, and the leader node determines the relocation destination by applying a hash function to the content.

Like content searching and caching, the leader node is also responsible for estimating content popularity, which helps determine which content is popular and provides benefit to the maximum number of consumers upon caching. To identify whether content is popular, DPCCache considers both local and global popularity. Each leader maintains a popularity table, as shown in Figure 5.1, to record the frequency of content

requests within its community, which represents the local popularity of the content. To obtain the global popularity of a requested content item, leader nodes exchange their local popularity information with other leaders, as shown by the dashed blue lines in Figure 5.1.

These operations of `DPCCache` are explained in detail in the following four subsections. First, the topology partitioning method used to form communities is described, followed by the processes of content searching and caching, and finally the method used for popularity estimation.

### 5.3.3 Identifying Neighborhood Router Group

In order to create a router group, we view the network as a graph with routers as nodes and connectivity between them as edges as defined in Definition 1. Further, we partition the network graph $G$ into several groups. Each group has a subset of nodes and edges as defined in Definition 2, and identify a set of communities in $G$ such that no two nodes are part of two communities as in Definition 3 and shown in Algorithm 5.1.

**Algorithm 5.1** Network Graph Partitioning

1: $C \leftarrow \text{Communities}(G)$
2: **for** $C_i \in C$ **do**
3: $\quad H_i \leftarrow \text{LeaderNode}(C_i)$
4: $\quad \text{Index}(C_i) \leftarrow \{\}$
5: **end for**

A set of routers to which a consumer is directly connected is known as edge routers, as defined in Definition 4.

**Definition 1.** *A network graph $G = (V, E)$ where $V$ is the set of nodes and $E$ is the set of links connecting nodes. An edge $e_i \in E$ is a pair of two vertices $(v_i, v_j)$ with $v_i, v_j \in V$.*

**Definition 2.** *Let $G$ = ($V$, $E$) be a network and a community $C$=($V(C), E(C)$) has $V(C) \subseteq V$ of vertices and $E(C) \subseteq E$ edges such that $\forall$ $e_i$=($v_i, v_j$) $\in E(C)$ with $v_i, v_j \in V(C)$*

**Definition 3.** *A community structure $C = \{C_1, C_2, \cdots C_K\}$ of a network $G$ = ($V$, $E$) is a partition of $V$. In other words, $\sum_{l=1}^{K} C_l = V$ and $C_i \cap C_j = \emptyset$ for $i, j = 1, ..., K$ with $i \neq j$. $K$ refers to the number of communities in $C$.*

**Definition 4.** *A router $R_e \in R$ is an edge router if $e_k$=($R_e$, $CM_j$) for a consumer $CM_j$ and $e_k \in E$.*

We adopt the well-known network graph partitioning algorithm (Louvain algorithm) as proposed in [21] to partition the large network topologies into community structures. Louvain algorithm is a heuristic approach to identify the communities, and it uses the modularity [109] optimization technique. Modularity is a metric used to assess the quality of communities formed during network partitioning. The algorithm identifies a set of communities from a network in two phases. The first phase of the algorithm is known as modularity optimization, in which the algorithm attempts to merge every node with its neighbor nodes with a high modularity score in order to form a better community structure. The second phase of the algorithm is known as community aggregation, in which the algorithm creates a new community network by aggregating communities discovered in the first phase. These two phases are iterated until the modularity score is positive. The steps involved in the algorithm for finding communities are summarized below.

**Step 1:** Initially, for the graph $G = (V, E)$, every node belongs to its own community. As a result, the number of communities of $G$ equals the number of nodes in $G$, *i.e.,* $K = V$.

**Step 2:** After forming $K$ communities, the algorithm begins selecting nodes one by one and attempts to merge node $R_i$ with the set of its neighboring nodes based on the modularity score.

**Step 3:** Node $R_i$ is moved from its current community $C_i$ to the neighboring community $C_j$ that yields the maximum positive modularity gain. If no positive modularity

gain is obtained, $R_i$ remains in its original community.

**Step 4:** Repeat Step 3 until the modularity score increases while moving the node from one community to another.

**Step 5:** Once no further gain in modularity is possible, the detected communities are aggregated to form a new graph $G'$, where each community is represented as a single node.

**Step 6:** The modularity optimization process (Steps 2–4) is reapplied to the aggregated graph $G'$.

**Step 7:** The algorithm terminates with a set of $K$ communities when no further increase in modularity can be achieved.

***Leader Node Selection:*** After forming the $K$ communities, we choose the node with the highest degree of centrality within each community as the leader node. It allows easy access to information with minimal overhead within the community. The leader node acts as a centralized entity responsible for managing the content popularity. It is also involved in content searching and caching within the community. We represent the leader node of any community $C_i$ as

$$H_i = \max_{v \in V} \{deg(v)\} \tag{5.1}$$

where $i \in 1, 2, \cdots k$, $H_i$ is the leader node of community $C_i$, $v$ is the node of community $C_i$, and $deg(v)$ denotes the degree of node $v$, *i.e.*, the number of neighbors of node $v$. For example, in Figure 5.1, Router $R_4$ has the highest degree of centrality in the community $C_1$, so it is elected as the leader of community $C_1$. Similarly, $R_{10}$ and $R_5$ are elected as the leaders of communities $C_2$ and $C_3$, respectively. Figure 5.2 illustrates the process of partitioning the graph into communities in the network topology $G$ and selecting a leader node $H_i$ for each community.

### 5.3.4 Handling Content Requests and Caching Decisions

The routers within a community cooperate to maximize the cache diversity. The leader node is responsible for making caching decisions for all the content and also for all other nodes within the community. Algorithm 5.2 outlines the procedure for

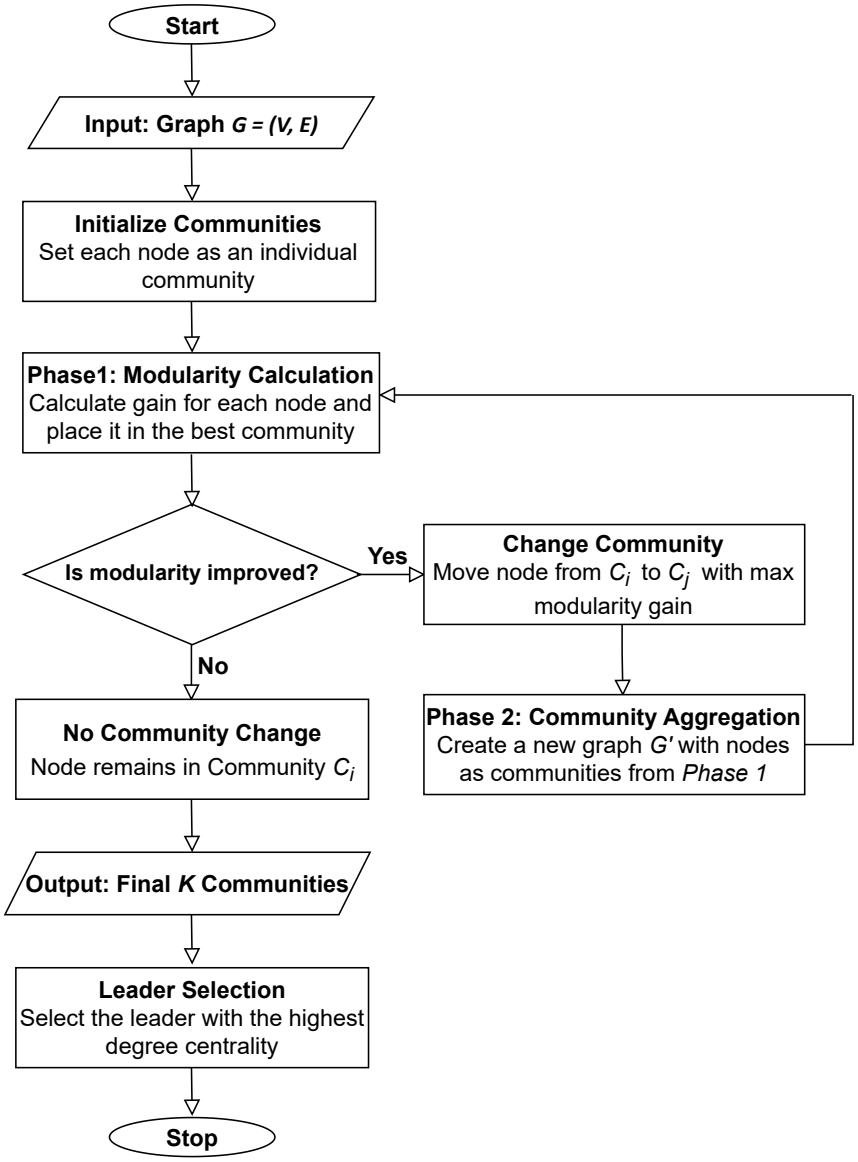



Figure 5.2: Community Formation and Leader Selection.

handling a content request.

When a new request arrives at a router $R_i$, it checks its internal cache for the content, and if available, it will serve the content from there. If the content is not available at $R_i$, then other routers within the same community as $R_i$ will be queried for the content. If none of the other routers have the content, then the request is

**Algorithm 5.2** Processing Content Request at $R_i$

```
1: Name ← New Content Request at R_i
2: if Name ∈ CacheOf(R_i) then
3:    Serve from R_i
4: else if R_i ∈ C_i then
5:    if Name ∈ CacheOf(R_j ∈ C_i) with R_i ≠ R_j then
6:       Serve from R_j
7:    end if
8: else
9:    Forward Content Request to Next Hop Router on the Shortest Path to Source
10: end if
```

forwarded to the next hop-on-path router. It is worth noting that this node may or may not be within the community. We make a few optimizations for this search by routing the query for content to other nodes within the community through the leader node. This has several advantages, as it reduces the repeated queries to other nodes in the community as request flows through the nodes of a community. Second, it helps to avoid the same content getting cached at more than one router's cache within a community. Typically, in the downstream direction, each router on the content delivery path between the content provider and the consumer decides whether or not to cache content. In contrast to traditional on-path caching techniques, the proposed strategy requires on-path routers to first coordinate with the leader node of the respective community before making caching decisions in order to efficiently utilize the cache space and prevent content redundancy within the community. Algorithm 5.3 describes the caching operation.

When new content arrives at a router $R_i$, the decision to cache the content is also made in consultation with the leader node $H_i$ within its community $C_i$. The name of the new content is shared with the leader node, which then decides whether the content can be cached or not at the requested Router. The leader node $H_i$ checks whether the content with the same name is cached elsewhere within the community

**Algorithm 5.3** Caching the content within community $C_i$

```
1: Name ← New Content at R_i
2: Size ← SizeOf(Name)
3: if Name ∉ CacheOf(R_i ∈ C_i) then
4:    if T_i − O_i ≥ Size then
5:       Cache Name at R_i      /* Maximize cache utilization */
6:       Update Index(C_i)
7:    else if R_i is an Edge Router then
8:       Cache Name at R_i      /* Use cache replacement policy (Algorithm 5.4) */
9:       Update Index(C_i)
10:   else
11:      P_Name ← PopularityOf(C_i, Name)
12:      Cache Name at R_i if P_Name ≥ δ   /* Use cache replacement policy (Algorithm 5.4) */
13:      Update Index(C_i)
14:   end if
15: end if
```

at any other router. If yes, then it discards the cache request for $R_i$; otherwise, the content can be cached at the router, provided the capacity of the content store (CS) is adequate to cache the content. Content popularity affects the way the content is cached within the community routers, especially when the CS capacity is full. To track the content popularity, we define a content counter `Index(`$C_i$`)` that tracks the most popular content within the community $C_i$. The proposed strategy keeps track of the most popular content within the community by establishing communication between the leader node and the other routers in the community. The `DPCCache` caching algorithm makes decisions differently depending on whether cache space is available or the cache is full.

*Router CS capacity is not full:* When router $R_i$ has enough space in its CS (*i.e.,* the currently available free cache space given by $T_i$; total cache space minus the $O_i$ occupied cache space), then the content is cached irrespective of its popularity. This

is motivated by the fact that DPCCache aims to maximize cache utilization and reduce upstream traffic towards the original content source by caching both popular and non-popular content. Subsequently, the popularity counter for the corresponding content is updated in the popularity table of community leader $H_i$.

*Router CS capacity is full:* When the CS is full at $R_i$, which is an edge router directly connected to the content requester, then $R_i$ evicts the existing contents using the LRU policy to make room for the new content and cache it. The key concept here is that caching content at the edge router reduces traffic and latency. After caching the new content, information will be updated in the popularity table of $H_i$. When the popularity level of new content $P_{Name}$ within the community $C_i$ reaches the popularity threshold $\delta$, router $R_i$ caches content $P_{Name}$ in its CS using the LRU policy. Following that, `Index(`$C_i$`)` for content `Name` will be updated in the popularity table of community leader $H_i$.

If the new content is not in any of the router's caches then only the content is cached within the community. The content is cached at router $R_i$ if there is space available to store it. This will maximize the utilization of the available cache. If the router $R_i$ is an edge router, then the content is cached. Otherwise, it uses a popularity estimation technique to decide whether the content needs to be cached or otherwise. If the content is popular, then it is cached at $R_i$. We elaborate on how the popularity of content is estimated in the next subsection.

### 5.3.5 Cache Content Replacement

When the router's CS capacity is full, and the popularity of newly arrived content exceeds the threshold value, the router chooses to evict older content from its CS to store the new content. The content evicted from the router's CS may be further requested by the consumers subsequently. So, the idea here is to give the evicted content a second chance by re-caching it within the community in order to improve the cache hit ratio and reduce the content retrieval time. DPCCache uses a hash-based content placement strategy to cache evicted content again within the community. The reason for using a hash-based approach is that the leader node quickly selects an appropriate

router to place the evicted content without requiring further coordination. Suppose some content has to be evicted from a router $R_i$. The router sends a message to the community leader node containing the content name. Upon receiving the message, the leader node computes a hash value $h$ by passing the content name to a hash function $f()$, which results in a numeric value. The leader node then applies a modulo operation to $h$ to identify another router $R_j$ ($R_i \neq R_j$) within the same community to re-cache the evicted content. The router $R_j$ may or may not cache the suggested content in its CS, depending on the availability of space in its own cache. Algorithm 5.4 illustrates how the leader node redirects content evicted from one router to be cached on another within the community.

**Algorithm 5.4** Content Eviction at a Router $R_i$

1: `EName` ← Select *Name* with LRU
2: `ESize` ← SizeOf(`EName`)
3: CommunitySize ← SizeOf($C_i$) − 1
4: `h` ← f(`EName`)
5: $R_X$ ← `h` % CommunitySize
6: **if** $T_X - O_X \geq$ `ESize` **then**
7: Cache Evicted Content `EName` at $R_X$; $R_X \in C_i$
8: **else**
9: Discard the Content `EName`
10: **end if**

For example, consider the reference architecture shown in Figure 5.1. When new popular content *NName* arrives at router $R_1$, the old content *EName* must be removed from $R_1$ to make room for content *NName*. Router $R_1$ sends evicted content *EName* to the leader node $R_4$, which calculates the hash value of *EName*; after calculating the hash value, suppose that $R_3$ is selected as the appropriate router to cache the *EName*, then leader node $R_4$ sends *EName* to $R_3$ to store in their cache. If there is enough space in $R_3$, it caches the *EName*; otherwise, it ignores the suggestion.

### 5.3.6 Content Popularity Estimation

As discussed in Chapter 4, caching decisions guided by content popularity improve the utilization of limited cache capacity. How popular a content item is determined by the number of times it is requested by consumers. Similar to the previous chapter, consumer request patterns are modeled using a Zipf-like content popularity distribution [23, 153], which characterizes the skewed nature of content access frequencies, as shown in Equation 5.2.

$$P(f) = \frac{\left(\sum_{m=1}^{M} \frac{1}{m^{\alpha}}\right)^{-1}}{f^{\alpha}}, f = 1, 2, ...., M \tag{5.2}$$

where $P(f)$ represents the probability of demand for the $f^{th}$ most popular content, $f$ is the rank of the content, $\alpha$ is the Zipf exponent or skewness parameter, and $M$ is the total number of distinct contents. When $\alpha > 1$ means content popularity is highly skewed, and the majority of consumers request content from a small set of popular content, while the requests for other less popular content start falling. When $\alpha < 1$, the popularity trend gradually starts falling, and when $\alpha$ becomes zero, the request pattern follows the uniform distribution, *i.e.*, all contents are equally requested by the consumers.

As mentioned previously, content popularity can be either local (estimated separately by individual routers) or global (by exchanging information about the popularity of different contents between routers). The latter approach of popularity estimation is better as it helps bring information about what is being cached among all the routers in a network. However, such a global popularity estimation significantly increases the load on the routers when the network size is large where every router in the network needs to exchange/send this information to every other router in the network. Further, this approach significantly hampers the scalability as the number of messages that needs to be exchanged increases exponentially with the addition of new routers and/or content types. Taking cognizance of this fact, we use a hybrid popularity estimation technique that combines the local and global popularity estimation as shown

in Equation 6.1.

$$P_{Name} = \eta \times P_{Local}(Name) + (1 - \eta) \times P_{Global}(Name) \tag{5.3}$$

where $P_{Name}$ is the popularity of content `Name` within a cluster, which is updated for every change in local and global content count values, $P_{Local}(Name)$ is the local popularity of the content `Name`, $P_{Global}(Name)$ is the global popularity of the content `Name`, and $\eta$ is the weight parameter in between 0 and 1. We set $\eta$ to 0.125, as empirically evaluated by [5, 7, 67], to give more weight to the global popularity count, helping capture global trends based on consumer interests from diverse regions and smoothing out short-term fluctuations. Algorithm 5.5 shows the approach followed in the estimation of content popularity within a cluster.

**Algorithm 5.5** Estimating the Popularity of Content

1: $P_{Global} \leftarrow 0$
2: **for** $C_j \in C$ and $C_j \neq C_i$ **do**
3: $\quad P_{Global} \leftarrow P_{Global} + P_{Name}$ from $C_j$
4: **end for**
5: $P_{Global} \leftarrow P_{Global}/(|C|-1)$
6: $P_{Name} \leftarrow$ Update Popularity as in Equation 6.1
7: Return $P_{Name}$

The main objective of our scheme is to estimate the content popularity in relation to the community while minimizing communication overhead. In the proposed popularity estimation technique, only the leader node of every community maintains a content popularity table alongside the CS, PIT, and FIB tables. This table acts as a repository of content information that helps to keep track of the popularity of content within the community and is used to make content caching decisions. The leader node collects the frequency of contents requested by the consumer(s) within the community and stores the content name along with their respective count value in the popularity table. For example, consider the reference topology shown in Figure 5.1, where routers $R_4$, $R_{10}$, and $R_5$ act as the leader nodes of communities $C_1$, $C_2$, and $C_3$, respectively. Each

leader node $R_4$, $R_{10}$, and $R_5$ maintains a popularity table to keep track of the popular contents. Table 5.1 shows the structure of the popularity table of leader nodes. To fetch the global frequency of the requested content, a leader node of community $C_i$ exchanges its local popularity (as estimated for community $C_i$) with the leader nodes of other communities ($C_j$'s $i \neq j$).

Table 5.1: Popularity Table

| **ContentName** | **Counter** |
|---|---|
| /video/Name1.mp4 | 2 |
| /document/Name2.pdf | 5 |
| /image/Name3.jpg | 5 |
| /video/Name4.mp4 | 7 |
| /video/Name5.mp4 | 10 |
| /video/Name6.mkv | 15 |
| /video/Name7.mkv | 20 |

`DPCCache` estimates content popularity on demand, meaning popularity is calculated only when needed. This approach enhances responsiveness to sudden spikes in content demand. By estimating popularity only for the requested content, `DPCCache` reduces the overhead of sharing unnecessary information. For example, when content `Name` is requested in the community $C_i$, its popularity is computed using Equation 6.1 by the community leader $H_i$ and then compared with a popularity threshold to determine whether it should be cached. The local popularity of `Name` refers to its popularity within $C_i$, while its global popularity is derived from information collected across multiple communities $C_j$.

To understand how many messages are reduced in the network and how much load is relieved from the routers when popularity estimation is performed through leader nodes, let us consider the example topology shown in Figure 5.3. The topology consists of two communities, $C1$ and $C2$, with their respective leaders, $LC1$ and $LC2$.

*Without communities and leader nodes:* Every router maintains its own local popularity table. The topology in Figure 5.3 has 9 routers, so all 9 routers maintain separate

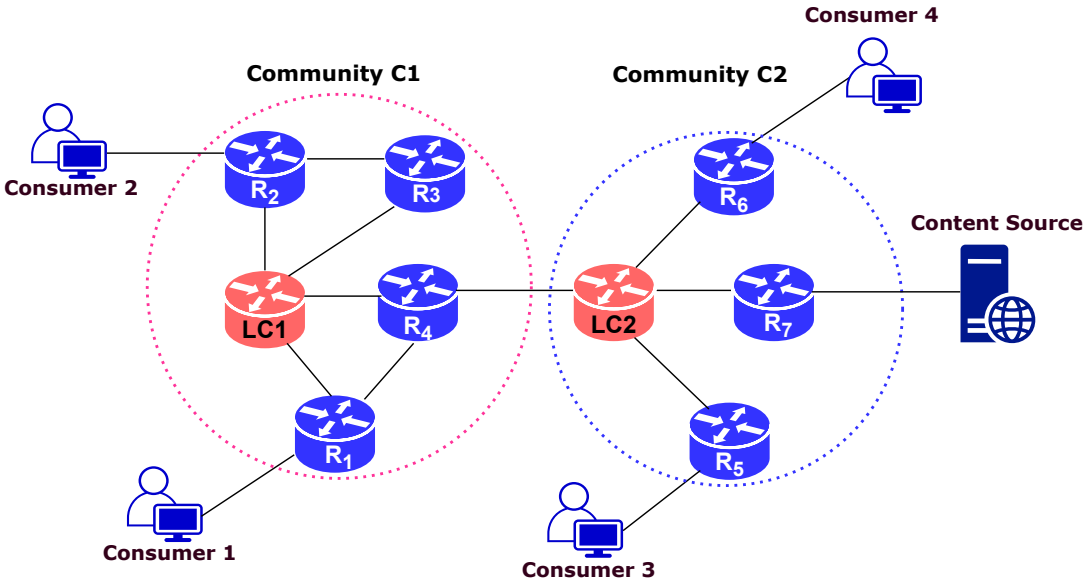



Figure 5.3: Example Topology with Two Communities.

tables. For global popularity estimation, each router sends its popularity information to every other router, and these messages travel through several intermediate nodes. This results in a total of 72 $(n \times (n-1))$ message exchanges. This shows that without leader nodes, the communication overhead for popularity estimation becomes very high, and the cost of maintaining the popularity table also increases. If there are $N$ routers in the network, and each router maintains a popularity table of size $s$ bytes, then the total storage overhead across all routers is $N \times s$ bytes.

*With communities and leader nodes:* The topology is divided into two communities, $C1$ and $C2$, with leader nodes $LC1$ and $LC2$ elected for each. Popularity estimation is now handled only by these two leaders, so only two tables are maintained, one at $LC1$ and one at $LC2$. In each community, all routers communicate only with their leader. In community $C1$, routers $R_1$, $R_2$, $R_3$, and $R_4$ send their information to $LC1$. In the worst case, community $C1$ requires only 4 messages to obtain local counts, as all four routers send their local popularity information to $LC1$. Similarly, in the community $C2$, routers $R_5$, $R_6$, and $R_7$ send their information to $LC2$, requiring 3 messages to obtain local counts. Thus, obtaining local popularity requires only 7 messages. For global popularity estimation, the two leaders exchange their local summaries, which

requires just 2 additional messages ($LC1 \leftrightarrow LC2$). In total, only 9 messages are needed, which is a significant reduction compared to the 72 messages without leader nodes. Also, the storage overhead is reduced from $N \times s$ bytes to $M \times s$ bytes, where $M$ is the number of leader nodes. With 2 leader nodes, the total storage overhead becomes $2 \times s$ bytes.

*Demand-driven popularity estimation:* `DPCCache` uses a demand-driven popularity estimation approach, which reduces the amount of information exchanged among peers. In traditional popularity estimation methods, even items that are rarely requested remain in the popularity table, and sharing the entire table increases both communication load and storage overhead. Moreover, some content details are not needed by leaders of other communities. For example, assume Table 5.1 is the popularity table of leader $LC1$, and it is shared with $LC2$. If some content items, such as `Name1` and `Name3`, are not of interest to $LC2$, exchanging this unnecessary information increases the load. Demand-driven estimation avoids this issue by exchanging information only for content that is currently requested and can adapt to capture frequent changes in content request patterns.

## 5.4 Performance Evaluation

This section outlines the simulation configuration, network topologies, and a discussion of simulation outcomes.

### 5.4.1 Simulation Setup

As in the case of `eNCache` and `PeNCache`, here we also use the Icarus [135] simulator for the performance analysis of `DPCCache`. We used the stationary traffic workload available in the Icarus to generate content requests. This workload is suitable for generating a large number of requests, similar to a real-world network traffic scenario. In the simulation, the arrival of content requests follows a Poisson distribution with a mean rate parameter of $\lambda = 10$ content requests per second. This means that, on average, 10 content requests are expected during a one-second interval. Similar to the

works [31, 50], we also use the Zipf distribution [23] as the content popularity model. The workload for each simulation experiment contains a catalog of ten thousand ($10^4$) distinct content objects evenly distributed across the content sources. We first generated ten thousand ($10^4$) requests to warm up the cache, which is helpful for reducing experimental errors. Cached contents during the warm-up period were not considered for the evaluation. After the warm-up period, to evaluate the performance of caching strategies, we generated one hundred thousand ($10^5$) content requests using the Zipf distribution model by varying the $\alpha$ value. In Icarus, the shortest path is computed using Dijkstra's algorithm to route the Interest and Data packets.

All caching strategies use the Least Recently Used (LRU) algorithm for replacement to make room for the new content when the cache space becomes full, since it performs cache lookup and replacement tasks quickly [7]. We set the Zipf parameter $\alpha$ between 0.6 and 1.4 for simulations, and cache stacks were installed in all routers in the network topologies with homogeneous cache sizes ranging between 0.05% and 0.3% of the catalog size (total content population), which is significantly smaller than the total contents available in the catalog. We fixed the popularity threshold $\delta$ value for the `DPCCache` and MPC at five to distinguish between popular and unpopular content. The reason for using a fixed $\delta$ value is twofold: i) to make a fair comparison with the MPC strategy, and ii) it enables the leader node to quickly distinguish between popular and unpopular contents rather than adjusting the $\delta$ value every time the content popularity (number of requests for a content) increases. The $\eta$ value for `DPCCache` in Equation 6.1 is set to 0.125, as sensitivity analysis conducted with different values of $\eta$ showed that 0.125 yields slightly better results compared to the other values. The results shown in all graphs represent averages over ten runs, with a 95% confidence interval. The configuration parameters for our simulation experiments are shown in Table 5.2.

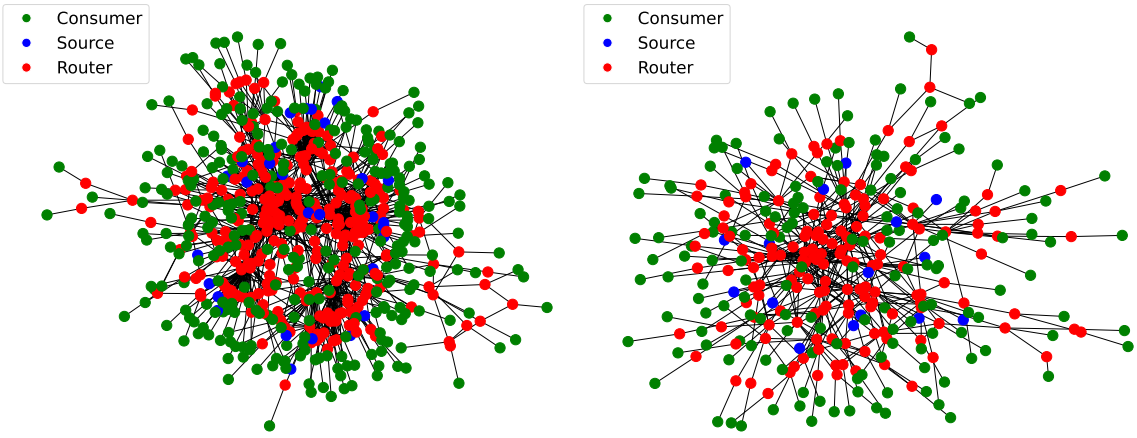
### 5.4.2 Network Topology Setup

For performance evaluation, similar to the caching strategies `eNCache` and `PeNCache`, `DPCCache` is also evaluated using the large-scale RocketFuel Internet Ser-

Table 5.2: Simulation Parameter Settings

| Parameters | Value |
|---|---|
| Network topology | RocketFuel ISP topologies |
| Content catalog size | $10^4$ objects |
| Content requests | $10^5$ objects |
| Cache warm-up | $10^4$ objects |
| Content request arrival rate | Poisson distribution, $\lambda = 10$ requests/sec |
| Content popularity models | Zipf distribution ($\alpha \in [0.6, 0.8, 1.0, 1.2, 1.4]$) and Uniform distribution |
| Routers cache capacity | Homogeneous, [0.05–0.30]% of catalog size |
| Content placement | Uniform distribution |
| Replacement policy | LRU |

vice Provider (ISP) topologies [146, 147], namely Sprintlink (US) and Tiscali (Europe), with autonomous system (AS) numbers AS 1239 and AS 3257, respectively. These two ISP topologies differ in both size and geographical region. Figure 5.4 shows the ISP maps of the AS 1239 and AS 3257 topologies, where consumer nodes are shown in green, source nodes in blue, and router nodes in red.




(a) AS 1239 (b) AS 3257

Figure 5.4: RocketFuel ISP Topologies: AS 1239 and AS 3257.

A similar process to the one used in Chapter 3 is followed here to create the

network topologies from the ISP map files. Here also, we use the default RocketFuel topology settings provided by the Icarus simulator [135]. The details of these two ISP topologies are presented in Table 5.3. To form the communities of these two topologies, we use the community generation algorithm [21], which is implemented in Python and integrated with the topology generator. Accordingly, the Sprintlink (AS 1239) topology is divided into 13 communities, and the Tiscali (AS 3257) topology into 11 communities.

Table 5.3: Details of Network Topologies

| **Topology** | **Node** | **Edge** | **Consumer** | **Source** | **Router** | **Router Degree** (Min, Max, Avg) | **Diameter** |
|---|---|---|---|---|---|---|---|
| Sprintlink (US) AS 1239 | 661 | 1318 | 315 | 31 | 315 | (2, 47, 7.26) | 136 |
| Tiscali (Europe) AS 3257 | 338 | 505 | 161 | 16 | 161 | (2, 31, 5.17) | 83 |

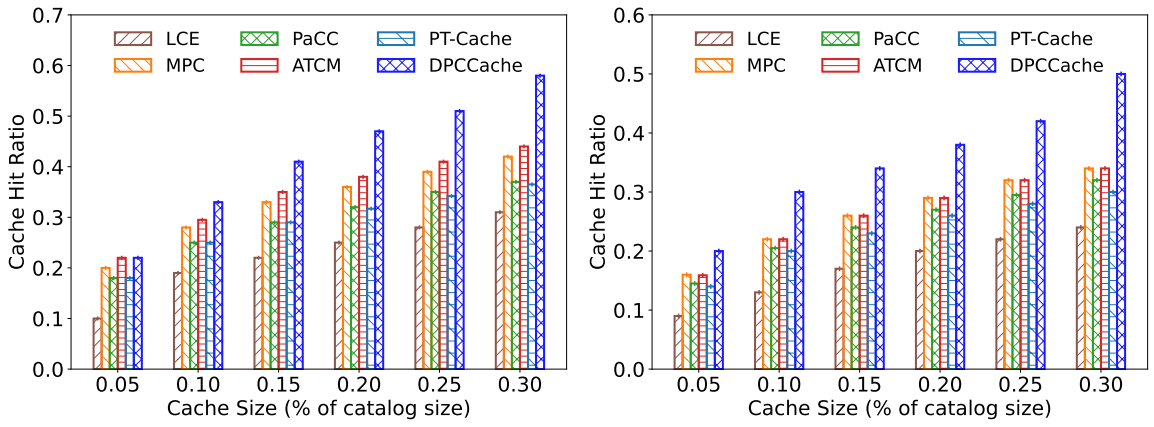
### 5.4.3 Simulation Results and Observations

Here, we present the evaluation results of DPCCache on these ISP topologies. We also compare its performance with other schemes and benchmarks. For comparison, we used classical caching methods such as LCE [84] and MPC [19], along with recent methods including PaCC [8], ATCM [124], and PT-Cache [77]. As in Chapter 3, we use performance metrics such as cache hit ratio, content access time, cache diversity, and an additional server workload metric for evaluation purposes. This server workload metric represents the fraction of content served by the original content source(s). A lower server workload reduces the burden on the original content source and prevents network congestion.

***Impact of Varying the Cache Sizes:*** Here, we discuss the impact of cache size on various performance metrics. The Zipf distribution's $\alpha$ parameter is set to a default value of 0.8 for this evaluation. Many studies [31, 118, 182] have suggested that using $\alpha$ value close to 1 captures the pattern of moderately popular content requests. When $\alpha$ is low, the request distribution is nearly uniform, whereas a high $\alpha$ results in a

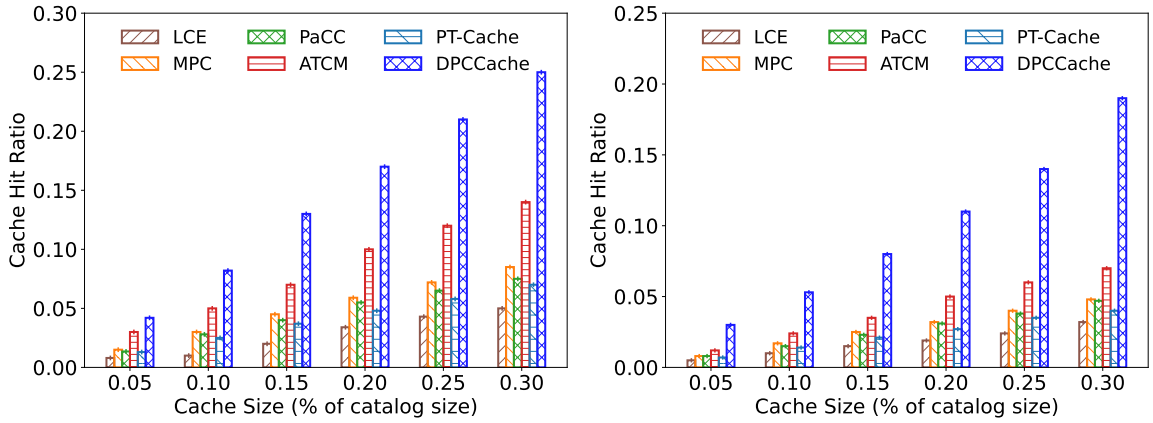
highly skewed pattern where a few content items dominate the requests. To study the impact of this on the performance, we set the cache capacity to 0.05%, 0.10%, 0.15%, 0.20%, 0.25%, and 0.30% of the catalog size (*i.e.*, total content population). These cache sizes are relatively small compared to the catalog size (*i.e.*, 10000).

Figure 5.5 shows the comparison of the cache hit ratio of `DPCCache` with other caching methods. It can be noticed that with the increasing cache capacity, the cache hit ratio of all caching strategies increases significantly. When compared to the other five caching schemes, `DPCCache` has a higher cache hit ratio regardless of the cache size. This significantly higher cache hit ratio of the `DPCCache` can be attributed to the fact that it does content searching and caching operations cooperatively, which results in efficient utilization of the limited storage capacity at each router. It is also worth noting that routers making cooperative decisions (ATCM) outperform other approaches that do not incorporate cooperation, such as LCE, MPC, PaCC, and PT-Cache. Figures 5.5a and 5.5b show that when the cache size is relatively small (*i.e.*, 0.05%), `DPCCache` improves the cache hit ratio up to ∼2.19 times on AS 1239 topology and up to ∼2.23 times on AS 3257 compared to other strategies. Further, when the cache size becomes large (*i.e.*, 0.30%), we get the improvements up to ∼1.9 times on AS 1239 and up to ∼2.1 times on AS 3257.




(a) AS 1239 (b) AS 3257

Figure 5.5: Cache Hit Ratio for Varying Cache Sizes across Different Network Topologies ($\alpha = 0.8$).



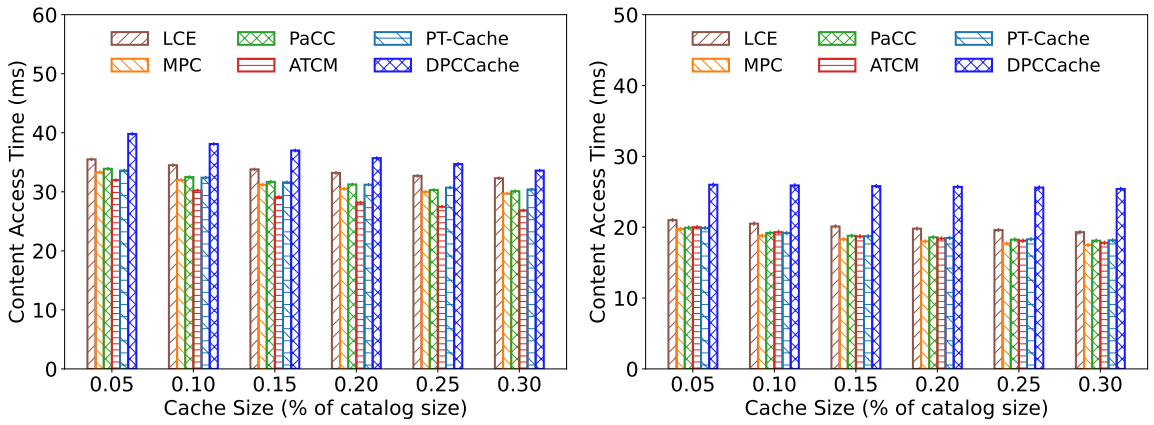

(a) AS 1239 (b) AS 3257

Figure 5.6: Cache Hit Ratio for Varying Cache Sizes across Different Network Topologies (Uniform Distribution, $\alpha = 0$).

Figure 5.6 shows the cache hit ratio of all six caching strategies for the two standard network topologies in the case of uniform content distribution pattern (*i.e.*, $\alpha = 0$). It is worth noting that when the consumer request pattern follows a uniform distribution, the cache hit ratio of all caching strategies decreases. Because the popularity of each requested content is equal, caching strategies need to perform cache and replacement operations multiple times, resulting in inefficient use of the limited cache capacity. Figures 5.6a and 5.6b show that when the cache size is set to 0.05%, `DPCCache` scheme improves the cache hit ratio on AS 1239 topology up to ∼5.3 times and on AS 3257 up to ∼6 times, whereas when the cache size is set to 0.30%, the improvements on cache hit ratio is up to ∼5 times on AS 1239 and up to ∼5.9 times on AS 3257. Figures 5.5 and 5.6 depict that `DPCCache` outperforms the other five caching strategies across both topologies, regardless of the request patterns and cache sizes.

Figure 5.7 compares the average content access time for retrieving requested content from both in-network caches and the original content source. Content access time of all caching strategies decreases as the cache size increases. Content access time of `DPCCache` is up to 19.1% higher on AS 1239 topology and up to 26% higher on AS 3257 than other popularity-based caching strategies, such as MPC, PaCC, ATCM, and PT-Cache. The reason for the higher content access time of `DPCCache` is due to the

delay caused by coordination between the router and the community's leader node. The location of the leader node inside the community also influences this delay; if the leader node is located far inside the community, it may take longer for the router to exchange the information.



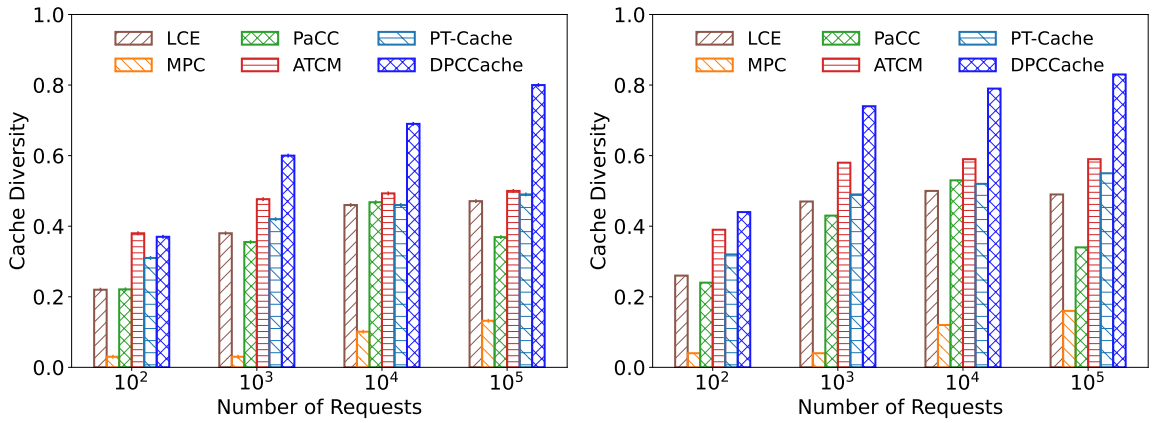

(a) AS 1239 (b) AS 3257

Figure 5.7: Content Access Time for Different Cache Sizes on Different Network Topologies ($\alpha$=0.8).

***Impact of Varying the Number of Requests:*** Here, we present the results for two performance metrics: cache diversity and server workload. These two values are measured at periodic intervals (*i.e.*, after $10^2$, $10^3$, $10^4$, and $10^5$ content requests) based on the number of requests generated by consumers during simulations. The reason for measuring these metrics after a certain number of requests is to observe how much unique content exists across the network relative to the total cached content, and to assess the load on the server once certain consumer requests have been generated. For these experiments, the Zipf distribution $\alpha$ is set to 0.8 and the cache size to 0.1% of the total content population. The reason for choosing the smaller cache size is that routers have very limited cache capacity in real network topologies.

Figure 5.8 shows the proportion of unique content to total available content in the network for all six caching strategies across both topologies. Figures 5.8a and 5.8b show that the `DPCCache` strategy cached up to 80% unique content on AS 1239 topology and up to 83% unique content on AS 3257. It is worth noting that when the

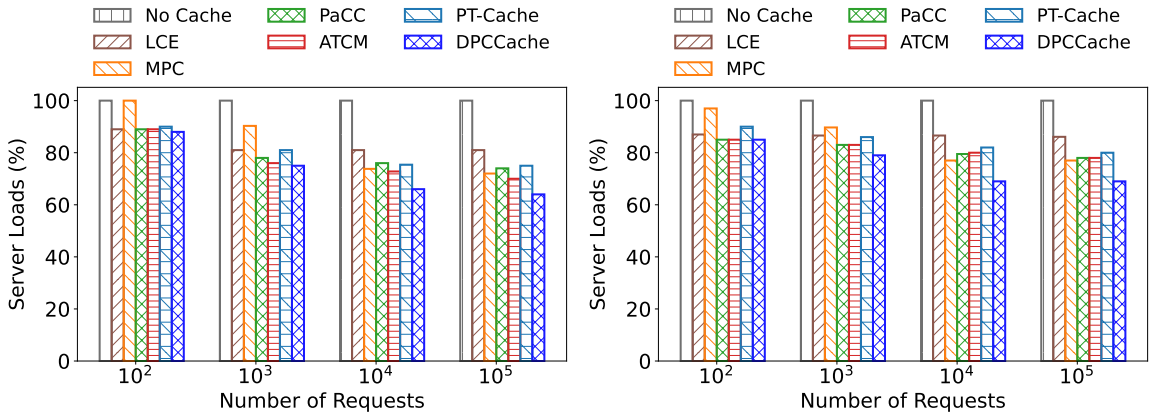
number of requests is low, the cache diversity of DPCCache is close to ATCM. However, when the number of requests increase (the most common scenario in realistic networks), DPCCache outperforms the rest. The reason for the higher cache diversity of DPCCache is due to cooperative content caching within the community, which minimizes content redundancy in the community.




(a) AS 1239 (b) AS 3257

Figure 5.8: Cache Diversity Comparison for Different Network Topologies, with Cache Size= 0.1% and $\alpha$= 0.8.

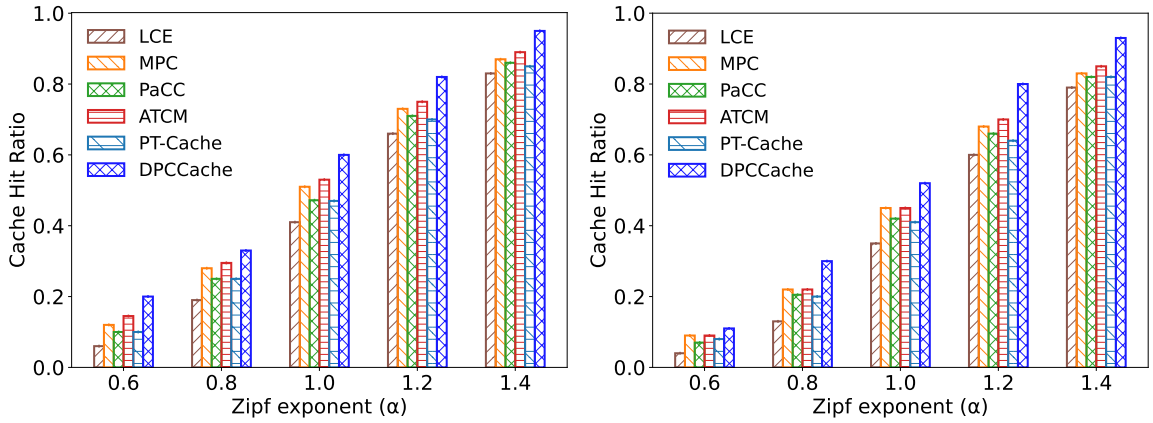
Figure 5.9 demonstrates the load on the server by showing the portion of requests the original content source served out of the total consumer-generated requests. No Cache strategy represents all the contents served from the original content source during the simulations (*i.e.*, 100% load on the server). We implemented the No Cache strategy in the simulation to observe how much the load on the server is reduced by caching techniques, compared to when no cache is available in the network. Figures 5.9a and 5.9b show that at the start of the experiments, the load on the server is approximately 90% for all caching strategies due to the limited contents available in the intermediate router's cache. When the number of requests increases, DPCCache reduces the load on the original content sources by up to 36% on the AS 1239 topology and up to 32% on AS 3257, compared to the scenario without caching (No Cache). The community division approach of DPCCache helps store the most popular content in the routers' caches, thereby reducing the load on the original content source.




(a) AS 1239 (b) AS 3257

Figure 5.9: Server Workload Comparison for Different Network Topologies, with Cache Size= 0.1% and $\alpha$= 0.8.

***Impact of Varying the Popularity Parameter*** $\alpha$***:*** Here, we discuss how the $\alpha$ value affects the cache hit ratio for various caching strategies across two different network topologies. The Zipf distribution governs content requests, with requests for popular content varying according to the skewness parameter $\alpha$. For the simulations, we use $\alpha = 0.6, 0.8, 1.0, 1.2,$ and 1.4. Figure 5.10 illustrates the impact of $\alpha$ on the cache hit ratio for all six caching strategies with a fixed cache size of 0.1%. From Figure 5.10, it can be seen that increasing the $\alpha$ parameter significantly increases the cache hit ratio of all six caching strategies. This is because as the $\alpha$ value increases, the possibility of the most frequently accessed content being stored in the network cache also increases. Figure 5.10a shows that on AS 1239 topology, `DPCCache` improves the cache hit ratio by up to ∼3.3 times compared to the other caching techniques at lower $\alpha$ (*i.e.*, $\alpha$= 0.6) and up to ∼1.14 times improvements at higher $\alpha$ (*i.e.*, $\alpha$= 1.4). Similarly, Figure 5.10b shows that on AS 3257 topology, `DPCCache` improves the cache hit ratio up to ∼2.8 times compared to the other caching techniques at lower $\alpha$ (*i.e.*, $\alpha$= 0.6) and up to ∼1.2 times improvements at higher $\alpha$ (*i.e.*, $\alpha$= 1.4). The margin at higher $\alpha$ values is smaller compared to lower $\alpha$ values. This is because a few contents are heavily requested when the $\alpha$ value is high, while others are requested less frequently, leading to an improved cache hit ratio for all caching techniques.



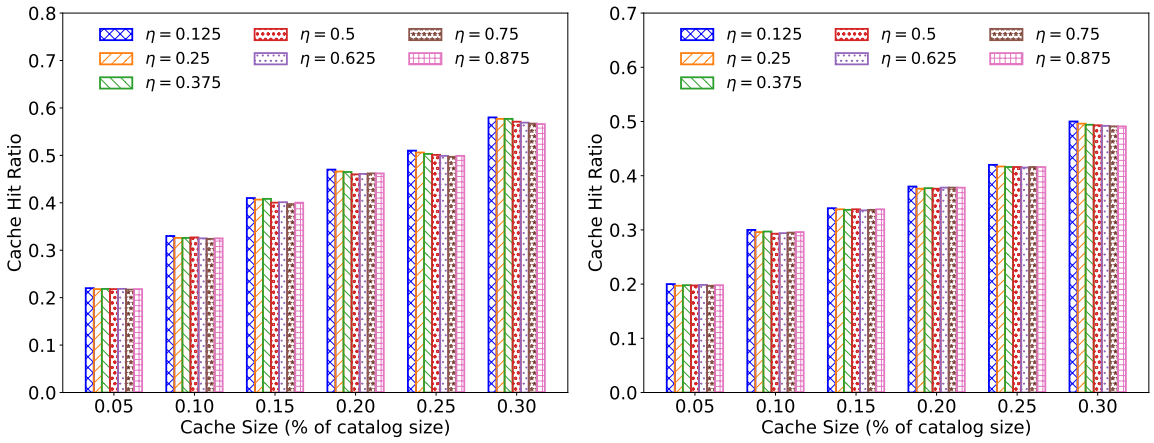

(a) AS 1239 (b) AS 3257

Figure 5.10: Cache Hit Ratio for Varying $\alpha$ Values Across Different Network Topologies (Cache Size = 0.1%).

***Performance of DPCCache with Different $\eta$ Values:*** DPCCache estimates the popularity of content by adjusting the value of the $\eta$ parameter in Equation 6.1. A lower $\eta$ value assigns more weight to the global request count, while a higher one assigns less weight. When making caching decisions, DPCCache considers both local and global content frequencies. Due to the cooperative nature of DPCCache, higher weight is given to global counts because caching content that is popular across communities benefits consumers in other communities as well. Here, we present a performance analysis study of DPCCache by changing the $\eta$ value. Through the simulation results shown in Figure 5.11, we can observe that, on both topologies, changing the $\eta$ value has little impact on the cache hit ratio. However, a $\eta$ value of 0.125 slightly performs better than the other values.

### 5.4.4 Analysis of Communication Overhead

It is worth noting that the community structure minimizes the communication overhead by restricting the global popularity estimation to only between leader nodes of a community. However, the caching and replacement decisions within the community are made by the leaders within the community. Thus, we measure the number of local and global message exchanges in the simulation experiment. Here, we measure




(a) AS 1239 (b) AS 3257

Figure 5.11: Effect of $\eta$ on Cache Hit Ratio in Different Network Topologies ($\alpha$= 0.8).

the communication overhead as a function of the number of communities/clusters in the network. To help in manually generating the desired number of clusters (the community detection algorithm generates clusters based on the structure of the graph), we use a small artificially generated network topology in this case. The topology used is shown in Figure 5.12.

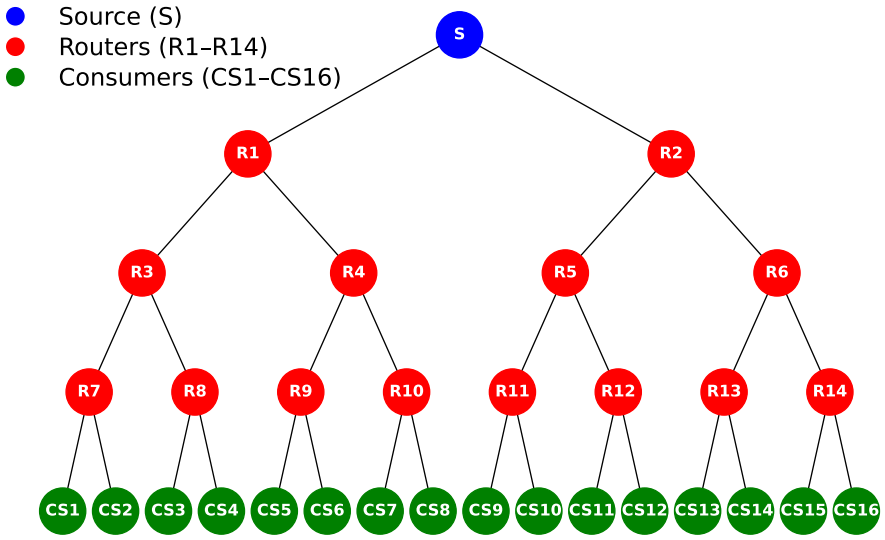



Figure 5.12: 5-Level Binary Tree Topology.

This is a 5-level binary tree topology consisting of 31 nodes with 16 leaf nodes as consumers (green), 14 internal nodes as routers (red), and a root node as a content

source (blue). In this experiment with this topology, the following simulation parameters were used: content population (content catalog) = $10^3$, content requests = $10^4$, cache size = 1% of the content population, and Zipf parameter $\alpha = 0.8$. The network topology shown in Figure 5.12 was manually divided into communities of varying sizes and subsequently measured the local and global messages exchanged during the simulations. The number of communities (K) is set to 1, 2, 5, and 10.

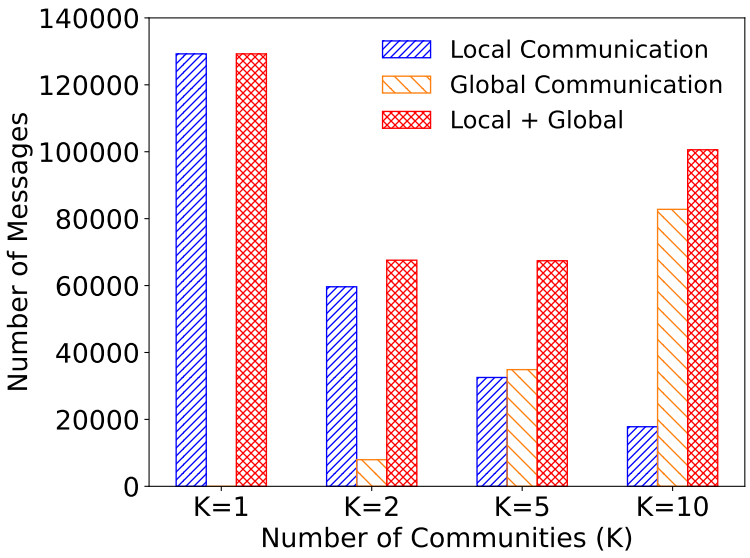
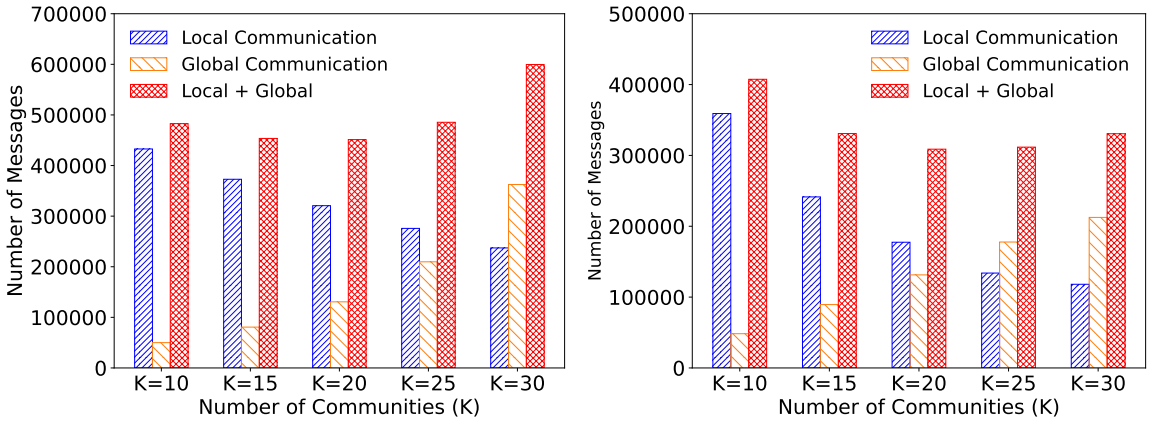



Figure 5.13: Total Number of Messages Exchanged vs. Number of Communities in the Network.

Figure 5.13 shows the number of local and global message comparisons for communities of varying sizes. From Figure 5.13, we can notice that when the entire network topology is considered as a single community (*i.e.*, K=1), the total number of local message exchanges is too high due to a large number of nodes in the community and no global communication is needed in this case. When the K value increases, local messages begin to decrease gradually while global messages increase. When the network topology is divided into ten communities (*i.e.*, K=10), the number of messages exchanged within the community (local messages) decreases while the number of messages exchanged outside the community increases. This is because the number of nodes within communities is reduced, resulting in a lesser number of messages exchanged within communities, but this burdens global coordination. In general, Fig-

ure 5.13 reveals that a smaller number of communities results in high communication overhead within the communities and low overhead outside the communities, whereas a large number of communities results in low network overhead within the communities and high traffic outside the communities. An optimal number of communities is required to balance these two overhead parameters. In Figure 5.13, the red bars indicate the sum of local and global messages. It can be observed that for $K = 2$ to 5, the total number of messages exchanged is lower compared to $K = 1$ and $K = 10$. However, at $K = 5$, there is a balance between local and global messages, making it the optimal community size.




(a) AS 1239 (b) AS 3257

Figure 5.14: Number of Messages Exchanged vs. Number of Communities for AS 1239 and AS 3257 Topologies.

We also conducted experiments on real-world network topologies (AS 1239 and AS 3257) using the same simulation setup employed to obtain the results shown in Figure 5.13 for the artificially created tree topology. Figures 5.14a and 5.14b present the overhead analysis for network topologies AS 1239 and AS 3257, with $K$ ranging from 10 to 30. For these experiments, communities are formed using the Louvain community detection method by tuning the resolution parameter to obtain the desired number of communities. Our observations are consistent with those from the 5-level binary tree topology. As the number of communities increases, the size of each community decreases, which leads to fewer local messages exchanged within communities

and a higher number of global messages across the network.

## 5.5 Handling Contents of Different Sizes

It is worth noting that in the results shown previously, each data chunk is assumed to be of the same size. However, in practice, different data items will be of different sizes. In order to assess the performance of `DPCCache`, which can indeed handle data chunks of varying sizes, this experiment is conducted. Handling data chunks of varying sizes requires changes to the content replacement algorithm. In this section, we furnish the details of the changed data chunk replacement algorithm and evaluation. Here, we consider the content size in the range of 1 to 10 (this can be extended to any number of unique sizes and 10 unique sizes is just an experimental convenience) and every unique content is randomly assigned a size. As mentioned before, content sizes require adjustments both in updating the cache occupancy and also when new content has to be cached. This may require the eviction of multiple cached contents from the router if there is no room. For example, if a router $R_i$'s cache is full and a new content of size $S$ arrives which is popular, and the router decides to cache it, then two existing contents $N_1$ and $N_2$ may be evicted if the size of $N_1$ and $N_2$ are less than $S$ individually, but cumulatively it will be sufficient to accommodate the new content. Algorithm 5.6 outlines the process of replacing content of different sizes from the cache of $R_i$. To re-cache the evicted content $N_i$ within community $C_i$, follow Algorithm 5.4.

Similar to the previous experiments, here also the simulation catalog consists of $10^4$ different contents, and consumers make $10^5$ random content requests of varying sizes. The cache size of each router varies between 10 and 50 ($0.1\% - 0.5\%$ of Catalog Size) to analyze how the caching strategy accommodates content when dealing with smaller cache sizes. To make room for newly arriving content, all caching strategies employ the LRU policy. The simulation is performed on the same ISP topologies, as detailed in subsection 5.4.2.

Figure 5.15 shows the cache hit ratio of six caching techniques under different cache and content sizes. The cache hit ratio of all six caching methods slightly decreases

**Algorithm 5.6** Caching Content of Different Sizes

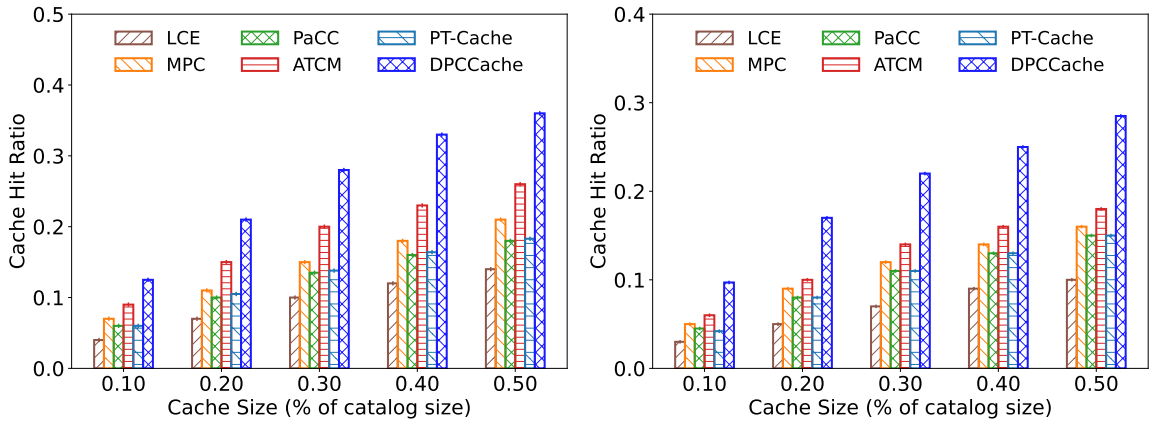
1: `Name` ← New Content at $R_i$
2: $N_i$ ← Evicted Content from CacheOf($R_i$)
3: $T_i$ ← Total Capacity of $R_i$
4: $O_i$ ← Current Occupancy of $R_i$
5: **if** `Name` $\notin$ CacheOf($R_i \in C_i$) **then**
6:  **if** $T_i - O_i \geq$ SizeOf(`Name`) **then**
7:  Cache `Name` at $R_i$ /* Maximize cache utilization */
8:  Update Index($C_i$)
9:  **else if** $R_i$ is an Edge Router **then**
10:  Cache `Name` at $R_i$
11:  Call ***ContentEviction***$(R_i, N_i)$ /* To make sufficient space */
12:  Update Index($C_i$)
13:  **else**
14:  $P_{Name} \leftarrow$ PopularityOf($C_i, Name$)
15:  Cache `Name` at $R_i$ if $P_{Name} \geq \delta$
16:  Call ***ContentEviction***$(R_i, N_i)$ /* To make sufficient space */
17:  Update Index($C_i$)
18:  **end if**
19: **end if**

**Function** *ContentEviction* $(R_i, N_i)$

1: **while** $(T_i - O_i) <$ SizeOf($Name$) **do**
2:  Evict $N_i$ from CacheOf($R_i$) /* Using cache replacement policy */
3:  $EvictedSize \leftarrow$ SizeOf($N_i$) /* Get the size of the evicted content */
4:  $O_i \leftarrow O_i - EvictedSize$ /* Update occupancy after eviction */
5: **end while**

as compared to the results reported in subsection 5.4.3, where each content was assumed to be of the same size (refer to Figure 5.5). The reason for the lower cache hit ratio is that, with varying content sizes, a router may need to evict multiple contents

from its cache to make room for a new arrival. The default caching technique (LCE) performs the worst since it caches every passing content, leading to the eviction of multiple contents to accommodate new content with a larger size. The ATCM [124] strategy performs better than the on-path techniques (LCE [84], MPC [19], PaCC [8], PT-Cache [77]) as it utilizes content available in routers outside the transmission path. However, from Figures 5.15a and 5.15b, it can be observed that even when considering varying content sizes, `DPCCache` outperforms the other five caching techniques. The superior performance of `DPCCache` is due to content searching, caching, and replacement being managed by the selected leader within the community, which ensures better utilization of the limited capacity of the routers. `DPCCache` achieves up to 43.47% higher cache hit ratio than ATCM (the second-best strategy) on the AS 1239 topology and up to 55.56% higher on the AS 3257 topology.

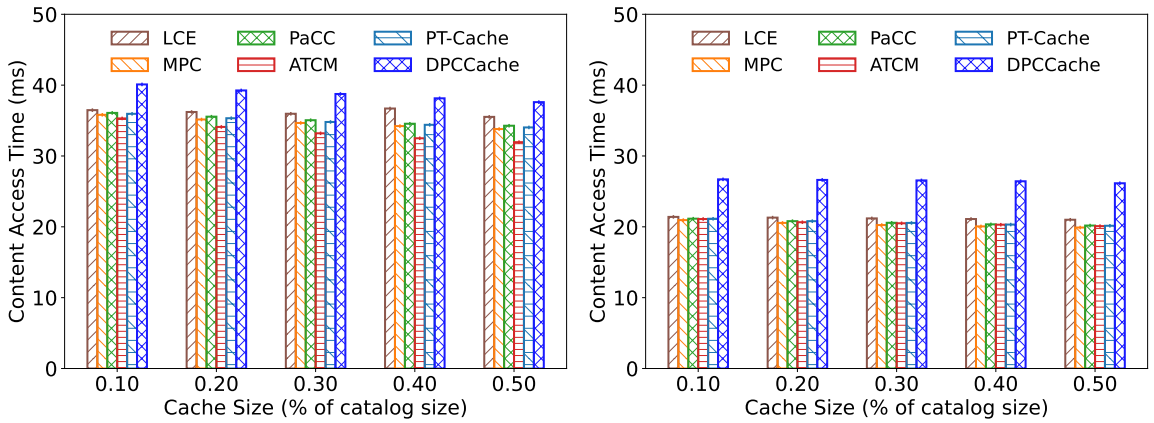



(a) AS 1239 (b) AS 3257

Figure 5.15: Cache Hit Ratio for Different Cache Sizes across Various Network Topologies with Varying Content Sizes.

Figure 5.16 demonstrates the average content access time of all six caching techniques under different content sizes. Content access time for all caching strategies decreases as the cache size increases because routers can accommodate more content. From Figures 5.16a and 5.16b, it can be seen that the content access time of `DPCCache` is higher than that of other caching schemes due to the router's need to search for content within the community through the leader node. The ATCM method has a

latency advantage because it directs requests based on information available from the neighborhood. However, this advantage comes at the cost of increased communication overhead, as routers must continuously exchange information about cache updates with neighbors.

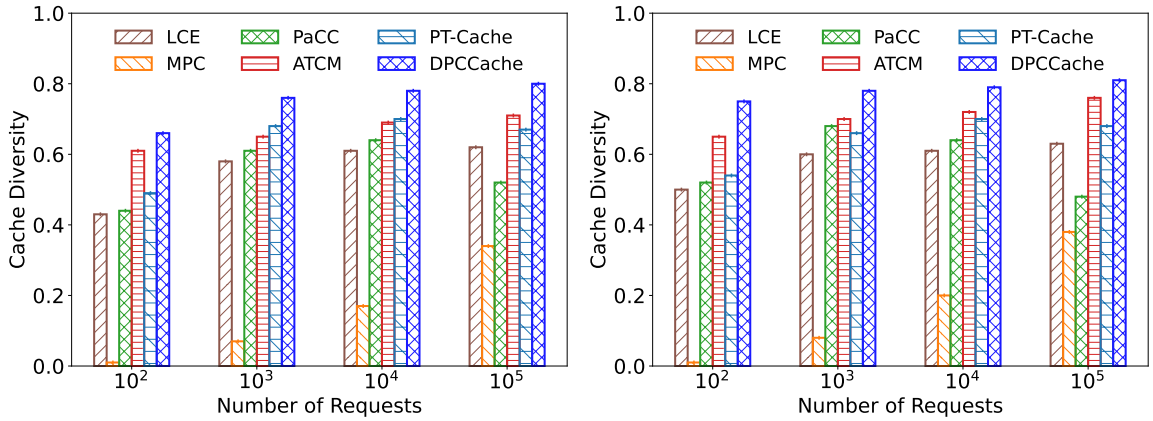



(a) AS 1239 (b) AS 3257

Figure 5.16: Content Access Time for Different Cache Sizes across Various Network Topologies with Varying Content Sizes.

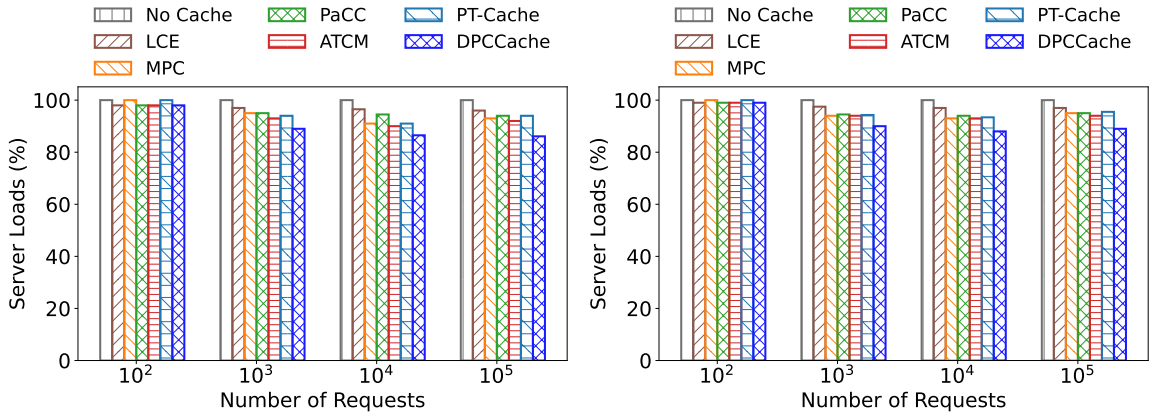
Similar to Subsection 5.4.3, here we also conduct simulations by varying content requests to evaluate cache diversity and server workload metrics. For these experiments, the number of content requests ranges from $10^2$ to $10^5$, while the cache size is fixed at 0.1% of the catalog size. Figure 5.17 shows the cache diversity of all six caching schemes across different network topologies. The cache diversity of `DPCCache` is higher than that of the other five caching techniques because `DPCCache` ensures no redundancy within the community. This observation highlights the benefits of making caching and replacement decisions in coordination with the leader node.

Figure 5.18 depicts the load on the original source resulting from each caching strategy across both topologies, with content of varying sizes. From Figures 5.18a and 5.18b, it can be seen that `DPCCache` imposes a lower load on the server compared to other caching techniques. This reduced workload on the server reflects that `DPCCache` better utilizes the network caches.




(a) AS 1239 (b) AS 3257

Figure 5.17: Cache Diversity Comparison for Different Network Topologies with Varying Content Sizes (Cache Size= 0.1%).




(a) AS 1239 (b) AS 3257

Figure 5.18: Server Workload Comparison for Different Network Topologies with Varying Content Sizes (Cache Size= 0.1%).

## 5.6 Discussion

DPCCache has been designed to achieve cooperation among routers for better utilization of the limited cache capacity. This cooperation extends to both caching and retrieving the content. Further, caching decisions are influenced by content popularity, which is estimated on demand by considering both local and global popularity. This

helps improve the cache hit ratio and diversity, and ultimately benefits consumers. The cooperative content searching and caching of DPCCache also helps reduce bandwidth consumption by satisfying most consumer requests from their local community.

DPCCache divides the large network topology into smaller communities to achieve cooperation. Cooperation in DPCCache follows a semi-decentralized approach in which a single leader is elected within each community to reduce communication overhead when coordinating with routers located outside the transmission path. The leader node of each community handles routing, caching, and popularity estimation operations, which helps reduce multiple queries to retrieve content and avoids caching the same content within the community. Although the proposed semi-decentralized scheme reduces communication overhead to some extent, it makes the caching scheme heavily reliant on the leader node [113].

Below, we discuss the limitations of DPCCache and potential solutions to overcome them:

*(i) Leader Node Placement:* In the current design, leader nodes are selected solely based on node connectivity. The node with the highest degree (*i.e.,* the maximum number of links) is elected as the leader of the community. However, the elected leader may not lie on the shortest path between a consumer and a provider, which can increase the time required for searching and caching content within the community. To address the challenge of content delivery latency, further exploration is required into selecting leaders using a weighted combination of factors such as node capacity, reachability to other nodes within the same community (intra-cluster distance), reachability to leaders of neighbouring communities (inter-cluster distance) [68], dynamic network conditions [172], etc., to make the method more robust.

*(ii) Leader Node Failure:* Leader node failure is another concern in DPCCache. The semi-decentralized nature of DPCCache creates a dependency on a single leader [113]. If the leader node fails, communication within the community is disrupted until a new leader is elected, which introduces delay. We foresee two potential methods to handle this. One method is to have a backup leader that can take over when the primary leader fails. In this approach, only one leader is active at a time, which avoids addi-

tional communication overhead, but it does not solve the load-distribution problem because all the workload still lies with a single active leader. The second method is to have multiple leaders within the community so that the workload is shared among them. Having multiple leaders [184] not only avoids communication failures caused by a single leader but also helps distribute the load in large-scale networks. Furthermore, the number of leaders within each community may vary depending on the community size. However, this approach becomes complex because multiple active leaders must be synchronized, which increases communication overhead due to additional coordination [184]. Further exploration is required to determine how to minimize this communication overhead and achieve balanced load distribution.

*(iii) Community Size:* Community size is another important factor. `DPCCache` uses static communities, meaning that once the topology is divided at the beginning, the community size remains fixed throughout the simulation. However, in real networks the community size may need to increase or decrease dynamically [24, 80] depending on changing network conditions. Forming communities based on factors such as similarity in content requests or the frequency of cache updates allows dynamic resizing [174, 175]. However, such dynamic community formation would lead to frequent changes and introduce significant overhead in maintaining updated information about which node belongs to which community and which leader it is associated with.

## 5.7 Summary

In this chapter, we described `DPCCache`, which is a community-based content popularity estimation and cooperative caching technique designed to reduce the communication overhead of popularity table exchange. `DPCCache` uses a community division algorithm to divide the large network topology into multiple communities to establish cooperation among the nodes. A leader node is elected within each community based on its connectivity with other nodes. The leader is responsible for popularity estimation, content searching, caching, and making replacement decisions. Simulation outcomes demonstrate that `DPCCache` outperformed other caching techniques in

terms of cache hit ratio, cache diversity, and server load, while slightly increasing access latency. The delay is due to content being searched through a community leader, which introduces additional time to the retrieval process. DPCCache simplifies both local and global popularity estimation and reduces the burden on routers other than the leader for maintaining popularity information. Furthermore, it minimizes content redundancy by caching only one copy of each content within the community. However, DPCCache estimates content popularity on an on-demand basis, which keeps the popularity table updated but introduces overhead on leaders. It also relies on a static threshold to distinguish between popular and non-popular content, which overlooks the dynamic nature of request changes over time. In the next chapter, we describe a community-centric cooperative technique where popularity estimation and caching decisions are made more effectively to minimize overhead.

# Chapter 6

# Predictive Popularity Caching for Efficient Content Delivery in NDN

## 6.1 Introduction

The content-popularity-based caching techniques (`PeNCache` and `DPCCache`) presented in the previous chapters show that incorporating popularity into caching decisions results in better utilization of router cache space. However, not considering the cache occupancy of routers during content placement may result in inefficient use of limited cache resources. This is because caching content without considering available space may result in keeping content that offers minimal value or causing unnecessary evictions. To achieve more effective cache utilization, this chapter initially presents a predictive caching model, called Predictive Popularity-based Caching (`PePC`), and later introduces an enhancement using a community-based approach, named Cooperative Predictive Popularity-based Caching (`CPePC`). In `PePC` and `CPePC`, the predictive model is designed to make caching decisions by considering the current cache occupancy of routers, inspired by the principles of the Random Early Detection (RED) [48] queue management technique. Both `PePC` and `CPePC` dynamically monitor cache occupancy and content popularity, using these factors in caching decisions to achieve more efficient and adaptive content placement. `PePC` allows each router to make its own decisions for request forwarding, caching, and popularity estimation.

The community-centric approach demonstrated in DPCCache shows that dividing the network topology into multiple communities not only improves the cache hit ratio but also simplifies cooperative decision-making for request routing, caching, and global popularity estimation. Building on these benefits, CPePC adopts a community-based design in which the network topology is partitioned into multiple communities, and each community elects a leader node to facilitate cooperation. This collaborative design in CPePC enhances routing, caching, and popularity estimation, improving overall content distribution. Similar to eNCache, CPePC also modifies the NDN packet structure [3] to support efficient searching and caching of content within the community. This modified packet structure reduces communication overhead between the leader and the other routers.

In summary, the key contributions of this chapter are as follows:

1. We introduce PePC and CPePC, novel caching techniques that make predictive caching decisions to improve the utilization of limited router storage across the network.

2. We design a predictive model to optimize router caching decisions, drawing inspiration from the principles of the Random Early Detection (RED) queue management technique.

3. PePC considers only the local popularity of content, with each router maintaining a popularity table to track the requests it receives.

4. CPePC adopts a community detection algorithm to partition the network topology into multiple communities or clusters, with a leader node designated in each community to coordinate cooperative routing, caching, and popularity estimation.

5. CPePC considers both local popularity (*i.e.,* content popular within a specific community) and global popularity (*i.e.,* content popular across all communities) when making caching decisions.

6. We implemented both PePC and CPePC algorithms in the Icarus simulator and compared their performance with several current NDN caching techniques, demonstrating their superior performance.

7. We discussed the challenges associated with deploying CPePC routing and caching algorithms in real-world NDN networks due to the cooperation required among routers.

The remainder of this chapter is structured as follows: Section 6.2 reviews related work in NDN caching that aligns with our proposed approaches. Section 6.3 presents the motivation and design details of the proposed PePC, while Section 6.4 presents the design details of CPePC. Section 6.5 describes the simulation setup and results. Section 6.6 discusses deployment considerations and highlights the limitations of PePC and CPePC. Finally, Section 6.7 concludes the chapter.

## 6.2 Prior Work

In this section, we review some notable existing works that are closely related to PePC and CPePC. As discussed previously, content searching, caching decisions, and popularity estimation can be performed either in a coordinated manner, in cooperation with other routers, or individually by each router. A more comprehensive analysis of these operations is provided in Chapter 2.

Cache storage is limited, and caching the same content at multiple routers increases content redundancy, which leads to inefficient utilization of resources. To overcome this, Max-Node Utility (MNU) [75] selects the best router(s) along the response path for content caching. This selection is based on the router's topological characteristics (betweenness and degree centrality) and dynamic characteristics (hop-distance from the requesting consumer and cache replacement ratio). Although MNU caches content at a few important routers in the network, it does not distinguish between popular and unpopular content. To improve cache efficiency in NDN, the authors of [125] proposed a method that prioritizes caching of reusable application data, such as video content,

by partitioning the cache storage into traffic classes.

Li *et al.* [90] designed popularity-driven caching schemes for NDN that minimize network traffic and content access time by determining optimal locations for content placement. Dynamic Popularity-Based Caching Permission (DPCP) [176] dynamically adjusts the threshold to cache popular content based on the number of content requests. Building on the idea of incorporating popularity, Popularity-aware Closeness-based Caching (PaCC) [7] considers both popularity and the router's closeness to consumers. PaCC caches content at routers that are topologically closer to the users. Another popularity-based caching technique, the Caching-Resource Utilization-Based Strategy (CRUS) [93], selects downstream routers with the lowest resource utilization for content caching. CRUS uses a probability threshold to filter popular content, where a higher threshold value increases the likelihood of caching the content closer to the consumer.

## 6.3 PePC: Design and Working Methodology

This section describes the operational workflow of the proposed PePC caching method, using the reference network topology shown in Figure 6.1. We begin with the design rationale, followed by the popularity estimation model, the content request forwarding process, and the caching decisions used in PePC. Table 6.1 summarizes the notations used throughout the description of PePC.

### 6.3.1 Design Rationale

The caching capability of NDN routers reduces content retrieval time and overall network traffic. However, due to limited cache size, it becomes crucial to determine which content should be cached, where it should be cached, and which content should be evicted to accommodate new arrivals. A carefully chosen caching technique, combined with an effective popularity estimation technique, enhances performance by caching frequently accessed content. However, solely caching popular content can lower the cache hit ratio [111], as it completely disregards the early access of less

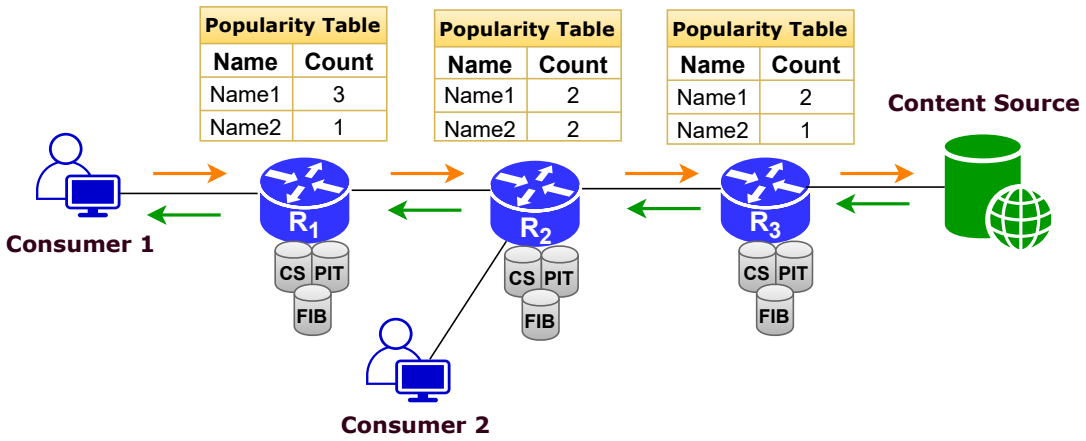



Figure 6.1: PePC Reference Architecture.

popular content. This may also lead to the under-utilization of existing cache space. On the other hand, caching every piece of content at all times does not optimize the overall performance of caching methods. Therefore, it is essential to carefully select which content should be placed in the router's cache, depending on its current occupancy level. To address this, we design a popularity-based early predictive caching (PePC) method, in which each router independently makes caching decisions based on its local cache occupancy and the popularity of incoming content relative to the content already stored. PePC dynamically adapts its caching behavior to maximize the cache hit ratio and minimize content access time. It avoids communication overhead, as routing, caching, and popularity estimation are all performed independently at each router.

### 6.3.2 Popularity Estimation Model

PePC uses content popularity to make caching decisions. Popularity estimation plays an important role in NDN networks, as it helps utilize cache space more effectively by keeping frequently requested content in the network. The popularity of a content item is determined by the number of times routers receive requests for it. A content is classified as popular when its request count reaches a static or dynamic threshold value, as discussed in Chapter 2. In real-world networks, content popu-

Table 6.1: Notations used in the Description of PePC and CPePC

| Notations | Description |
|---|---|
| $G$ | Network graph |
| $V$ | Set of nodes in the network |
| $R$ | Set of cache-enabled routers |
| $C$ | Set of communities |
| $N$ | Number of communities |
| $L$ | Set of leader nodes |
| $\lambda, \omega \in [0, 1]$ | Weight parameters |
| $P(Name)$ | Popularity of content `Name` |
| $L_f$, $G_f$ | Local and Global request counters |
| $PTable$ | Popularity table of $L_i$ |
| M | Number of distinct contents in $PTable$ |
| T | Period for popularity estimation |
| $\mathcal{A}_{R_i}$ | Average cache occupancy of $R_i$ |
| $\Theta_{R_i}$ | Current cache occupancy of $R_i$ |
| $\mathcal{RO}$ | Relative cache occupancy |
| $min_{th}, max_{th}$ | Minimum and maximum cache thresholds |
| $\rho_1, \rho_2 \in [0, 1]$ | Tunable parameters for cache thresholds |
| $S_{R_i}$ | Total storage capacity of the router $R_i$ |
| $P_1, P_2$ | Caching decision values |
| $P_{max}$ | Maximum value for $P_1$ |
| $\beta$ | Counter between cached and non-cached units |
| $\Delta$ | Popularity threshold |

larity varies over time with user demand. Therefore, an adaptive caching method should account for these changes by measuring popularity within a fixed interval `T` [79]. To measure popularity over time, we use the Exponential Weighted Moving Average (EWMA) [50, 98] model, which considers both present usage patterns and

**Algorithm 6.1** Updating Content Popularity at Router $R_i$

1: `Name` ← New Content Request at $R_i$
2: **if** `Name` ∈ PTableOf($R_i$) **then**
3: $LCount(Name)++$
4: **else**
5: Add `Name` into PTable
6: $LCount(Name) = 1$
7: **end if**

historical trends. Consumer content access patterns remain relatively stable over short time intervals, so frequently updating content popularity provides little benefit while increasing communication overhead due to information exchange and increasing content replacement rates. Each router in the network maintains an individual popularity table (PTable) to track content requests, as shown in Figure 6.1. This approach does not introduce communication overhead, since each router independently counts content requests without coordination. This local popularity estimation method ensures that each router tracks only the requests it receives. Each router maintains a variable called `LCount` (Local Counter) to track the content requests it receives. Upon receiving a request, the router checks if an entry for the requested content already exists in the PTable. If it does, the `LCount` value is incremented by one; otherwise, a new entry is created for the content, and `LCount` is initialized to 1. Consider the network topology shown in Figure 6.1, where each router $R_1$, $R_2$, and $R_3$ maintains a popularity table. For example, router $R_2$ receives requests for two different contents, `Name1` and `Name2`, each with an `LCount` of 2. Upon receiving a request for either `Name1` or `Name2`, the corresponding `LCount` is incremented by 1. If a request for a new content, say `Name3`, arrives, a new entry is created in the table with `LCount` initialized to 1. Algorithm 6.1 outlines the approach used to estimate the request count of each content in the PTable at router $R_i$. Equation 6.1 presents the moving average-based popularity estimation.

$$P_T(Name) = \lambda \times P_{T-1}(Name) + (1 - \lambda) \times LCount(Name) \tag{6.1}$$

Here, $P(Name)$ denotes the popularity of content `Name` at router $R_i$ over a time period $T$, $P_{T-1}(Name)$ represents the previous popularity of the content during the interval $T-1$, $LCount(Name)$ is the local frequency of content `Name`, and $\lambda \in [0,1]$ is the weighting factor of the EWMA. Using this popularity value, we derive the dynamic threshold, as given by Equation 6.2. The dynamic threshold represents the relative popularity of a newly arrived content at router $R_i$ with respect to all other contents cached at $R_i$.

$$\Delta_T = \frac{\sum_{k=1}^{M} P(Name_k)}{M} \tag{6.2}$$

Here, $\Delta_T$ denotes the dynamic popularity threshold, $P(Name_k)$ represents the popularity of content $Name_k$ during time `T`, and $M$ represents the total number of distinct contents in the PTable during that time interval. $\Delta_T$ is determined as the average number of requests per content the router receives during the time T. Contents are ranked as popular when they surpass the popularity threshold $\Delta_T$, which is updated periodically.

### 6.3.3 Interest Forwarding

Interest packets in `PePC` are forwarded towards the potential content provider (router or original source) following the default NDN on-path strategy. Each router processes the request relying on three tables (*i.e., CS, PIT, and FIB*). The FIB is built using name prefixes and the corresponding next best hop(s). Upon receiving an Interest, the router checks for the longest matching prefix and forwards the Interest to the appropriate interface. Algorithm 6.2 outlines the `PePC` Interest forwarding process. This on-path forwarding process does not add additional communication overhead, as routers do not coordinate with their neighbors. For example, consider the topology shown in Figure 6.1. When router $R_1$ receives a request from Consumer1, it forwards the Interest to the next hop $R_2$ if the content is not present in its cache. A similar operation continues at each subsequent router along the path until the matching Data packet is found in the cache in response to the corresponding Interest.

**Algorithm 6.2** Handling Content Request at $R_i$

```
1: Name ← New Content Request at R_i
2: if Name ∈ CacheOf(R_i) then
3:     Serve from R_i
4: else if InFaceOf(Name) ∈ PIT_{R_i} then
5:     Drop Interest        /* Incoming interface for Name already exists in PIT */
6: else
7:     Create PIT entry for InFaceOf(Name) at R_i
8:     Forward request to the next hop using FIB_{R_i}
9: end if
```

### 6.3.4 Predictive Caching Decisions

Here, we discuss how PePC decides to cache the content. This is a fundamental aspect of any caching scheme, as it determines how efficiently the limited cache space can be utilized and managed to maximize content availability in the network. This includes two major components: caching and replacement strategies. Caching decisions involve deciding which content to cache and where to cache it, while replacement strategies indicate which content to evict to make room for the new arrival.

***PePC Dynamic Content Placement Technique:*** The content caching decision of our proposed PePC strategy draws inspiration from the Random Early Detection (RED) [48] queue management technique. In order to reduce network congestion, RED uses a probabilistic decision-making approach to drop or forward packets as they arrive at a queue, based on queue occupancy by setting predefined values of minimum and maximum thresholds. Based on the RED principle, we design a predictive decision-making algorithm that determines the chance of caching at each router by considering the average cache occupancy. The average cache occupancy of the router helps in determining whether the router can cache the content or not, based on the current occupancy of the router. Our objective in making such a caching decision is to maximize the cache hit ratio and cache utilization. The average cache occupancy

of the router is updated using Equation 6.3.

$$\mathcal{A}_{R_i} = (1 - \omega) \cdot \mathcal{A}_{R_i,old} + \omega \cdot \Theta_{R_i} \tag{6.3}$$

where $\mathcal{A}_{R_i}$ represents the current average cache occupancy of $R_i$, $A_{R_i,old}$ represents the last updated cache occupancy value, $\omega$ represents the cache weight factor, and $\Theta_{R_i}$ represents the current cache occupancy of $R_i$. We use minimum ($min_{th}$) and maximum ($max_{th}$) thresholds for comparison with average cache occupancy in the caching decision process. These thresholds are determined based on the total capacity of the router, as illustrated in Equations 6.4 and 6.5. The $min_{th}$ and $max_{th}$ represent the reserved fractions of space allotted for content caching, reflecting the lower and upper bounds.

$$min_{th} = \rho_1 \cdot S_{R_i} \tag{6.4}$$

$$max_{th} = \rho_2 \cdot S_{R_i} \tag{6.5}$$

where $S_{R_i}$ represents the total storage capacity of the router $R_i$, while $\rho_1$ and $\rho_2$ are tunable parameters that range between 0 and 1 to adjust the minimum and maximum cache thresholds. These parameters are adapted based on content caching decision-making and the available storage capacity of the router.

Along the content forwarding path, each router independently makes caching decisions based on one of three conditions:

*Case 1: Average cache occupancy is less than the minimum threshold*

$$\mathcal{A}_{R_i} < min_{th} \tag{6.6}$$

According to the condition shown in Equation 6.6, $R_i$ caches all incoming contents regardless of their popularity in order to make the best use of the available cache space. It is worth noting that caching everything will not conflict with our objective of maximizing cache hit ratio and minimizing latency, as unpopular content will eventually be replaced with popular content as the occupancy increases.

*Case 2: Average cache occupancy is between the minimum and the maximum threshold*

$$min_{th} \leq \mathcal{A}_{R_i} < max_{th} \tag{6.7}$$

When the cache occupancy lies between the minimum and maximum thresholds, as indicated in Equation 6.7, $R_i$ caches the content only when the condition $\mathcal{RO} \geq P_2$ is satisfied; otherwise, the content is discarded. Here, $\mathcal{RO}$ represents the relative occupancy.

$$P_1 = \frac{P_{max}(\mathcal{A}_{R_i} - min_{th})}{(max_{th} - min_{th})} \tag{6.8}$$

$$P_2 = \frac{P_1}{1 - (\beta \cdot P_1)} \tag{6.9}$$

Equations 6.8 and 6.9 are used to calculate the early prediction parameters $P_1$ and $P_2$ for the content caching decisions. The value of $P_1$ ranges between 0 and $P_{max}$, where $P_{max} \in [0, 1]$ represents the upper limit for $P_1$, and $\beta$ denotes the number of content items that have not been cached since the last cached item. When content is not cached, the value of $\beta$ is reset to zero.

$$\mathcal{RO} = \frac{P(Name, R_i)}{max(P(R_i))} \tag{6.10}$$

Here, $\mathcal{RO}$ is a numerical value ranging from 0 to 1, which signifies the relative cache occupancy of $R_i$ calculated as in Equation 6.10. $P(Name, R_i)$ represents the popularity of content `Name` at $R_i$, and $max(P(R_i))$ denotes the maximum popularity value among all contents in the PTable at $R_i$.

*Case 3: Average cache occupancy reaches the maximum threshold:*

$$\mathcal{A}_{R_i} \geq max_{th} \tag{6.11}$$

When the average cache occupancy reaches the maximum threshold, as shown in Equation 6.11, the router stores only the most frequently accessed content to enhance content availability. Specifically, if the popularity score $P$(Name) exceeds the dynamic threshold $\Delta_T$, router $R_i$ caches the content. This approach helps improve the overall cache hit ratio in the network.

Algorithm 6.3 outlines the caching decision made by router $R_i$ upon receiving a new content `Name`. The steps outlined above are captured in the algorithm, which behaves differently depending on whether the average occupancy falls into one of three bins: below the minimum threshold, above the maximum threshold, or between the minimum and maximum thresholds.

**Algorithm 6.3** Caching Content at $R_i$

1: `Name` $\leftarrow$ New Content at $R_i$
2: Size $\leftarrow$ SizeOf( `Name`)
3: $S_{R_i} \leftarrow$ TotalCacheCapacityOf($R_i$)
4: $\Theta_{R_i} \leftarrow$ CurrentCacheOccupancyOf($R_i$)
5: **if** `Name` $\notin$ CacheOf($R_i$) **then**
6:  **if** $\mathcal{A}_{R_i} < min_{th}$ **then**
7:  Cache `Name` at $R_i$ /* Maximize cache utilization */
8:  Update $\Theta_{R_i}$
9:  **else if** $min_{th} \leq \mathcal{A}_{R_i} < max_{th}$ **then**
10:  Cache `Name` at $R_i$ if $\mathcal{RO} \geq P_2$
11:  Update $\Theta_{R_i}$
12:  **else**
13:  Cache `Name` at $R_i$ if $P(Name) \geq \Delta$
14:  Update $\Theta_{R_i}$
15:  **end if**
16: **end if**

Consider the network topology shown in Figure 6.1 to understand the caching algorithm of `PePC`. Assume that router $R_2$ caches 6 content items, each of equal size, with min$th$ = 2 and max$th$ = 4. Initially, the occupancy of $R_2$ is 0, as all 6 slots are available. When `Name1` arrives at $R_2$, it is cached according to Case 1, regardless of its popularity. The next content, `Name2`, also falls under Case 1 and is cached similarly. When `Name3` arrives, the cache occupancy is now greater than or equal to min$th$, so Case 2 applies. If the caching condition is met, `Name3` is cached. The same process applies to `Name4`, after which the cache occupancy reaches max$th$, *i.e.,* 4. Subsequently, `Name5` is handled according to Case 3, where it is cached only if it is considered popular; otherwise, it is discarded. Once the cache becomes full after filling all six slots, the replacement policy is used to replace less popular content with more popular content.

## 6.4 CPePC: Design and Working Methodology

In this section, we present an extended version of PePC, named CPePC, which incorporates the advantages of a community-driven approach and also global content popularity. We begin this section by providing the motivation behind the design of CPePC, followed by a description of how the native NDN packet structure is extended to support content search and caching. Then, we present the detailed design and relevant algorithms for community formation, local and global popularity estimation, request routing, and predictive caching decisions for content placement. We use the reference architecture of CPePC, shown in Figure 6.2, to describe and explain its overall design and operation. Table 6.1 summarizes the notations and symbols used throughout the description of CPePC.

### 6.4.1 Motivation

PePC dynamically adapts its caching decisions depending on the available cache space in the router. However, due to its non-cooperative nature, each router independently makes decisions for routing, caching, and popularity estimation. This can result in the same content being cached at multiple locations in the network, increasing content redundancy. PePC also fails to capture global content request patterns, as each router determines content popularity solely based on its local popularity table. Considering content popularity from a global perspective can benefit a larger number of consumers. Another limitation of PePC is that routers located on the request forwarding path are unaware of content availability outside the path. As shown with DPCCache, community-based approaches offer advantages by facilitating cooperation. Hence, in CPePC, we combine the benefits of community-based cooperation and popularity estimation. Similar to DPCCache, CPePC divides the network into communities, each of which elects a leader to manage intra-community and inter-community communication. Following the approach of PePC, CPePC also dynamically monitors cache occupancy and reacts accordingly, so that it starts caching popular content as cache occupancy increases.

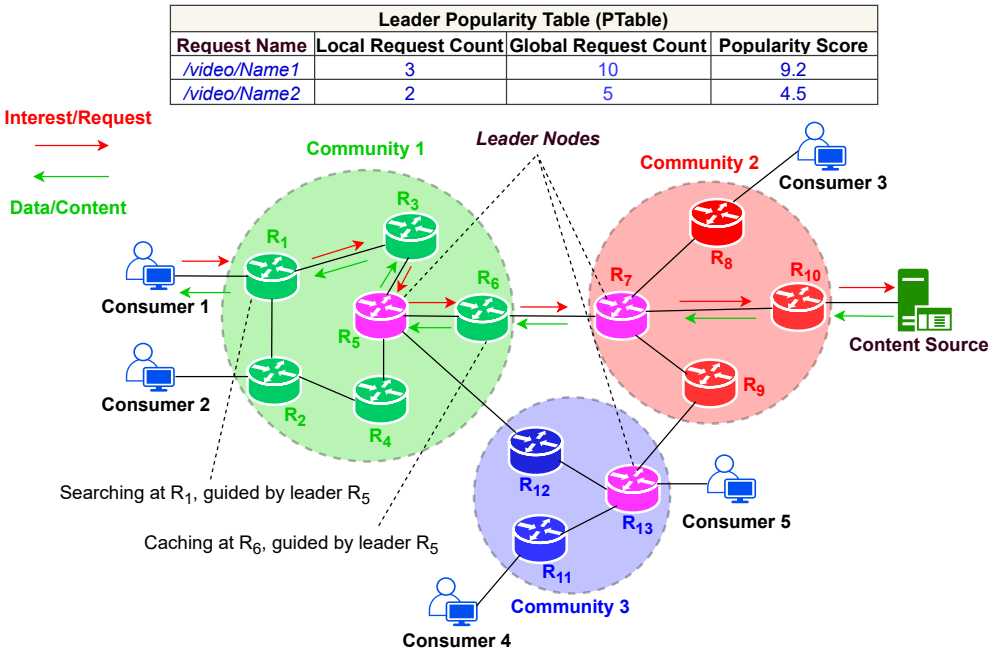



Figure 6.2: CPePC Reference Architecture.

### 6.4.2 CPePC Packet Structure

CPePC uses separate packets for communicating with the community leader and for forwarding requests towards the content provider using next-hop information. The Interest Packet Structure (IPS) is used to forward requests from one router to another router, while the Control Packet Structure (CPS) facilitates communication between a router and the community leader. We added a field to the native NDN Interest packet (IPS) format to maintain leader information. Additionally, we designed a new packet structure, CPS, to optimize content searching and caching within the community, while aligning with the standard NDN packet format. The fields marked in green, as shown in Figure 6.3, are newly added to the native NDN packet format specification (version 0.3) [3]. In CPS, the *Nonce* and *ForwardingHint* fields serve the same purpose as in the native NDN Interest packet format.

Below, we elaborate on the details of these extended fields.

- *ContentName:* The hierarchical name that uniquely identifies the content being requested or cached (e.g., `/video/Name`).

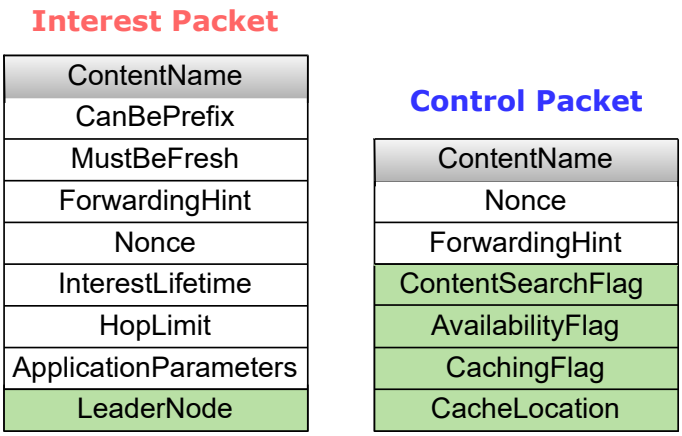



Figure 6.3: CPePC Interest Packet Structure (IPS) and Control Packet Structure (CPS).

- *Nonce:* Uniquely identifies a packet.
- *LeaderNode:* This field carries the names of the leader nodes. If a leader's name already exists, it is not added; otherwise, it is added to this field. For example, consider the topology shown in Figure 6.2. If Consumer1 requests content, the request first arrives at router $R_1$, where the *LeaderNode* field is initially empty. $R_1$ coordinates with the leader, *i.e.,* $R_5$, and when forwarding the packet to the next-hop $R_3$, $R_1$ adds $R_5$ to the *LeaderNode* field. Now, $R_3$ is part of the same community (*i.e.,* Community 1) as $R_1$, so $R_3$ forwards the packet to the next hop without modifying this field.
- *ForwardingHint:* Provides the forwarding path for content retrieval within the community.
- *ContentSearchFlag:* This field indicates whether the content should be searched within the community with the help of the leader. If the value of this field is set to `True`, the request is forwarded to the leader.
- *AvailabilityFlag:* Indicates whether the requested content is available within the community. If this flag is `True`, the leader responds to the router by setting the *ForwardingHint* field.
- *CachingFlag:* This field is set to `True` when the router receives a Data packet,

and the content is not available in its cache.

- *CacheLocation:* Specifies the location within the community where the content is to be cached.

### 6.4.3 Community Formation

In order to establish cooperation among the routers, we use a graph representation to divide large-scale network topology into several communities or clusters. In a network graph G, nodes typically represent source nodes (the original storage of contents), a set of consumer nodes (making content requests), and a set of routers capable of caching content, while the edges define the connections or links between these nodes. To form groups or communities, we use an approach similar to that in the `DPCCache` technique, using the Louvain algorithm [21], a well-known community detection method, to partition the network topology into multiple communities. The Louvain algorithm efficiently divides large networks into non-overlapping communities. An example network topology is shown in Figure 6.2, where three such communities are indicated.

This algorithm operates through two iterative phases, with the goal of improving the network's community structure by optimizing the modularity score. Modularity is used to assess the quality of the community structure within a network. The first phase of the algorithm is known as modularity optimization; in this phase, the algorithm begins with each node forming its own community in the network graph G. So, the number of communities in G equals the number of nodes in G. Then, it computes the modularity score and iteratively checks each node by calculating the gain in modularity score from joining the node from the nearby community $C_i$ to $C_j$. Nodes from community $C_i$ will only join community $C_j$ if it results in an increase in the modularity score. This phase terminates when no further gain in modularity can be achieved by moving the node from one to another. The algorithm's second phase is the community aggregation phase. In this phase, each community detected in the first phase is treated as a single node, forming a smaller network on which the

algorithm can be applied iteratively. For example, consider a network of six nodes $(R_1, R_2, R_3, R_4, R_5, R_6)$, where phase 1 yields three communities based on modularity score: $C_1 = \{R_1, R_2\}$, $C_2 = \{R_3, R_4, R_5\}$, and $C_3 = \{R_6\}$.

In phase 2, these communities are represented as three nodes: $C_1$ (Node1), $C_2$ (Node2), and $C_3$ (Node3). Modularity optimization is then performed on this new network, and if a gain in modularity is achieved, nodes are moved from one community to another. The process iterates between these two phases until no further improvement in modularity is achievable by combining communities. The algorithm terminates by creating the communities, and each node is assigned to one of the communities. A detailed discussion of the steps of the Louvain algorithm is provided in Chapter 5.

***Leader Node Selection:*** Following the division of the network graph G into N communities, we select a leader node for each community to facilitate cooperation among the communities. This node will be in charge of guiding the request packet, estimating the popularity of the content, and dictating the caching decisions for all the nodes within the community. We select the node with the highest betweenness centrality as the leader node within each community. The reason for choosing the node with the highest betweenness centrality as the community's leader is twofold: first, since the leader node lies on many shortest paths, it can easily reach other nodes in the community with fewer messages and also acts as a bridge between different communities of the network; second, if some node disconnects from the leader due to link failure, the leader can still establish communication with the affected node via an alternate path. If multiple nodes have the same number of betweenness centrality, the node with the minimum average distance from each node within the community is selected as the leader of that community. This approach of selecting a leader makes communication easier and reduces overhead. If the leader node of any community fails, the next node within the community with the highest betweenness centrality is elected as the new leader.

The leader node $L_i$ of any community $C_i$, $i \in \{1, 2, \ldots, N\}$, is selected as given in

Equation 6.12.

$$L_i = \underset{v \in C_i}{arg\,max}\{BC(v)\} \tag{6.12}$$

where $L_i$ represents the leader node of community $C_i$, and $BC(v)$ signifies the betweenness centrality of node $v$ within community $C_i$.

### 6.4.4 Popularity Estimation Model

Based on consumer usage patterns, content is classified as either locally popular or globally popular, as discussed in Chapter 2. If the content is popular in a particular geographical region or community, it is termed as locally popular, and if it is requested by a wide range of consumers from many geographic locations, it is considered globally popular. In order to optimize the limited router storage, it is important to consider both local and global popular content for caching decisions. Unlike `PePC`, `CPePC` takes into account both local and global request patterns when making caching decisions. To achieve this, the leader node of the community maintains a popularity table (PTable), which keeps track of all the content requests inside or outside the community. By maintaining this table only at the leader node, we reduce the burden on other routers to keep track of content popularity. Content popularity fluctuates over time within specific regions or communities, influenced by consumer interests. Similar to `PePC`, `CPePC` also periodically updates content popularity over a fixed time interval $T$ to adapt to these changes. For popularity estimation, the leader of each community maintains two variables, $L_f$ and $G_f$, which track the local and global frequency of requests for content `Name`. $L_{f_i}$ indicates the frequency of `Name` within a community $C_i$, and $L_{f_j}$ indicates the frequency of `Name` from community $C_j$ where $C_i \neq C_j$. $L_{f_i} \forall i \in C$ collectively represent the global popularity $G_f$ of the requested content `Name`. Algorithm 6.4 outlines the approach the leader node $L_i$ uses within each community $C_i$ to monitor the request count $L_f$ in the PTable. The PTable so computed by the Algorithm 6.4 is exchanged by leader nodes with other leaders every `T` time period, as shown in Algorithm 6.5. After exchanging tables, each leader node has the overall content requests ($G_f$) made during the `T` time interval for each content.

**Algorithm 6.4** Estimating Request Count at $L_i$

1: $L_f \leftarrow 0$ /* Initialization of the local request counter */
2: PTableOf($L_i$) $\leftarrow \{\}$ /* Initialization of the popularity table */
3: `Name` $\leftarrow$ Content Request in $C_i$
4: **for** each `Name` in Community $C_i$ **do**
5: **if** `Name` $\in$ PTableOf($L_i$) **then**
6: $L_f$(`Name`) $++$
7: Update PTableOf($L_i$)
8: **else**
9: Create Entry for `Name` in PTableOf($L_i$)
10: $L_f$(`Name`) $= 1$
11: Update PTableOf($L_i$)
12: **end if**
13: **end for**

When new content is requested, an entry is created in the PTable for that content request (lines 9 to 11 in Algorithm 6.4), and if an entry already exists in the PTable, the counter value is incremented by one (lines 5 to 7 in Algorithm 6.4). PTable stores four types of information: `Name` (content request name), $L_f$(`Name`) (count of content requests within community $C_i$), $G_f$(`Name`) (count of content requests across communities), and $P$(`Name`) (popularity score of the content, calculated using Equation 6.13). The structure of PTable is shown in Figure 6.2. The popularity of content $P$(`Name`) is calculated using Equation 6.13, which applies the EMWA approach.

$$P_T(Name) = \lambda \times P_{T-1}(Name) + (1 - \lambda) \times G_f(Name) \tag{6.13}$$

where $P_T(Name)$ denotes the popularity of content `Name` over the period `T`, $P_{T-1}(Name)$ represents the popularity score of content `Name` recorded in the PTable during the previous time step `T-1`, $G_f$ is the overall count of the number of requests for the content `Name` over the period `T`, and $\lambda \in [0, 1]$ is the EMWA smoothing factor.

For example, consider two content requests, `Name1` and `Name2`, requested by con-

sumers within the network. Assume that the previous popularity score of `Name1` is 3.5, and its global request frequency is 10. Using Equation 6.13 with $\lambda = 0.125$, the updated popularity score of `Name1` becomes 9.2. Similarly, for `Name2`, assuming a previous popularity score of 1.5 and a global frequency of 5, the updated popularity score becomes 4.5. The popularity score of `Name1` is higher than that of `Name2`, so `Name1` is considered more popular.

In order to adapt to changing consumer request patterns over time, we compute a dynamic popularity threshold $\Delta$ within each community instead of using a static threshold to classify content as popular or unpopular. Once the content reaches the threshold $\Delta$, it is considered popular. The dynamic popularity threshold $\Delta$ is computed over the time period `T`, as shown in Equation 6.14.

$$\Delta_T = \frac{\sum_{k=1}^{M} P(Name_k)}{M} \tag{6.14}$$

where $\Delta$ represents the dynamic popularity threshold of community $C_i$, $P(\texttt{Name}_k)$ represents the popularity of content $\texttt{Name}_k$ during time `T`, and $M$ represents the total number of distinct contents, $\texttt{Name}_1, \texttt{Name}_2, \ldots, \texttt{Name}_M$, in the PTable during that time interval.

### 6.4.5 Interest Forwarding

Unlike `PePC`, in `CPePC` the Interest forwarding is performed in a cooperative manner. Content searches within any community $C_i$ are conducted through the leader of the respective community. The content request handling procedure of the `CPePC` is shown in Algorithm 6.6. Whenever a router $R_i$ receives a request for content `Name`, it does the following: first, it checks its cache; if the content is available in its cache, it responds with the content. Otherwise, $R_i$ forwards the request to either the leader node of the community or the next hop router along the shortest path to the content source. We use the modified NDN packet structures, as shown in Figure 6.3, to forward requests within each community. The forwarding process is guided by including the leader's information in the *LeaderNode* field of the Interest Packet Structure (IPS),

**Algorithm 6.5** Estimating the Popularity of Content

1: $L_{f_i}(\texttt{Name}) \leftarrow$ Request Count of $C_i$
2: $L_{f_j}(\texttt{Name}) \leftarrow$ Request Count of $C_j$
3: **for** each $C_i \in C$ **do**
4: **for** each Time interval `T` **do**
5: Receive PTableOf($L_j$) from $C_j$
6: $G_f(\texttt{Name}) \leftarrow L_{f_i}(\texttt{Name}) + L_{f_j}(\texttt{Name})$
7: Update PTableOf($L_i$)
8: $P(Name) \leftarrow$ Update Popularity as in Equation 6.13
9: **end for**
10: **end for**
11: Return $P(Name)$

and by setting the *ContentSearchFlag* field to `True` or `False` in the Content Packet Structure (CPS). These modifications reduce the number of messages exchanged during content search within the community. Note that the CPS packet is used only for communication between the router and the community leader, while forwarding packets to the upstream router toward the content provider is done using the IPS.

Each router within the community knows its respective leader node. So, when router $R_i$ receives the request, it checks the *LeaderNode* field of IPS. If the leader node specified in the *LeaderNode* field matches the leader of $R_i$, $R_i$ can directly forward the request to the next upstream router (line 13 in Algorithm 6.6), as it means that the leader node has already been visited by the previous hop router within the same community as $R_i$. Otherwise, it forwards the request to the leader node by setting *ContentSearchFlag* to `True` and searches for the content within the community with the help of the leader (lines 4 to 10 in Algorithm 6.6). Upon receiving the request, the leader node responds with one of three types of messages: firstly, if the content is available in the leader's CS, it replies with content (Data packet); secondly, if the content is available in router $R_j$ (Router $R_j$ is in the same community as $R_i$) within community $C_i$, it replies with the location of $R_j$ (set the *ForwardingHint* field in CPS

**Algorithm 6.6** Handling Content Request at $R_i$

1: `Name` $\leftarrow$ New Content Request at $R_i \in C_i$
2: **if** `Name` $\in$ CacheOf($R_i$) **then**
3: Serve from $R_i$
4: **else if** Leader $L_i$ of $R_i \notin$ *LeaderNode* **then**
5: Forward Content Request to $L_i \in C_i$
6: **if** `Name` $\in$ CacheOf($L_i$) **then**
7: Serve from $L_i$
8: **else if** `Name` $\in$ CacheOf($R_j \in C_i$) with $R_i \neq R_j$ **then**
9: Serve from $R_j$
10: **end if**
11: **else**
12: *LeaderNode*.append($L_i$)
13: Forward Content Request to Next Hop Router on the Shortest Path to Source
14: **end if**

to specify the path an Interest packet follows to retrieve the Data); and thirdly, if the content is not available within the community, it responds with message NDATA (*AvailabilityFlag=False*) which Indicates that the requested content is unavailable within the specific community. When $R_i$ receives the second type of message, it routes the request to $R_j$ using the path specified by *ForwardingHint* to fetch the content. When $R_i$ receives the third type of message, it appends the visited leader node information to the *LeaderNode* field and routes the request to the next hop router by consulting the FIB. This procedure is performed in each community until the content is found. In `CPePC`, only the first entry router of each community is responsible for coordinating with the leader node during request forwarding, due to the *LeaderNode* field of IPS. This approach reduces communication overhead, as routers in the path do not need to coordinate with the leader. With the extended NDN packet structure, `CPePC` enables efficient content retrieval from both within and outside the community.

For example, consider the topology shown in Figure 6.2. When router $R_1$ receives

a content request from Consumer1, it checks the leader node *LeaderNode* field of the IPS. If it does not find a matching leader node, this means that the request for this Interest has not yet been sent to the leader. Therefore, $R_1$ sets *ContentSearchFlag* = `True` and sends the request to the leader $R_5$. Assuming the content `Name` is available in Community1 at router $R_4$, after receiving the reply from leader $R_5$, $R_1$ forwards it to $R_4$ via the path: $R_1 \rightarrow R_2 \rightarrow R_4$. If the content is not available in Community1, then the request is forwarded through the shortest path towards the content source by appending the leader information to *LeaderNode* field.

### 6.4.6 Caching Decisions

Similar to `PePC`, `CPePC` also makes caching decisions by considering both the cache occupancy of the router and the popularity of the content. However, in `PePC`, caching decisions are made independently by each router, which increases content redundancy, as multiple routers along the content delivery path may end up caching the same content. To reduce such redundancy, `CPePC` performs both content placement and content replacement operations collaboratively, with the support of the leader node in each community. `CPePC` ensures that content is not duplicated within a community; however, the same content may be cached in other communities. Similar to request routing, we have also optimized the caching process through communication with the community leader. In the downstream direction, when the content arrives at router $R_i$ within community $C_i$, $R_i$ forwards one copy towards the consumer to avoid delay caused by caching decisions, and another copy to its leader to potentially cache the content by setting the *CachingFlag* to `True` in the CPS, as shown in Figure 6.3. Note that if router $R_i$ already has the content, it will not forward a copy to the leader. Upon receiving the request, the leader node decides whether the content should be cached. If the content is to be cached, the leader sets the *CacheLocation* field to the selected router. This decision assumes that the leader is aware of the locations and cache occupancy levels of routers within the community through coordination. Algorithm 6.7 describes the content caching decision made by the leader node upon receiving a new content `Name`.

**Algorithm 6.7** Caching Content within Community $C_i$

1: `Name` $\leftarrow$ New Content at $R_i$
2: Size $\leftarrow$ SizeOf( `Name`)
3: $S_{R_i} \leftarrow$ TotalCacheCapacityOf($R_i$)
4: $\Theta_{R_i} \leftarrow$ CurrentCacheOccupancyOf($R_i$)
5: **if** `Name` $\notin$ CacheOf($R_i$) **then**
6: Forward Content `Name` to $L_i \in C_i$
7: **if** `Name` $\notin C_i$ **then**
8: **if** $\mathcal{A}_{R_i} < min_{th}$ **then**
9: Cache `Name` at $R_i$ /* Maximize cache utilization */
10: Update $\Theta_{R_i}$
11: **else if** $min_{th} \leq \mathcal{A}_{R_i} < max_{th}$ **then**
12: Cache `Name` at $R_i$ if $\mathcal{RO} \geq P_2$
13: Update $\Theta_{R_i}$
14: **else**
15: $P(Name) \leftarrow$ PopularityOf($C_i, Name$)
16: Cache `Name` at $R_i$ if $P(Name) \geq \Delta$
17: Update $\Theta_{R_i}$
18: **end if**
19: **end if**
20: **end if**

When content is unavailable within the community, the leader node either caches the content itself or instructs a selected router within the community to cache it if any of the following conditions are met; otherwise, the content is discarded.

(i) Average cache occupancy is below the minimum threshold (cache all content to maximize utilization).

(ii) Average cache occupancy is between the minimum and maximum thresholds (cache content based on the predictive model).

(iii) Average cache occupancy reaches the maximum threshold (cache only highly popular content).

These three caching conditions are explained in detail in the discussion of the caching decision algorithm of PePC. Unlike PePC, where each router independently considers these conditions to make caching decisions, in CPePC, the leader node performs this task for the entire community. To improve content response time, the leader selects a router that lies on the content delivery path between the provider and the consumer. This content placement scheme pushes content closer to the requesting consumers without introducing additional computational overhead for router selection within the community. Note that the leader node may also cache the content, because it is a cache-enabled node, if it is located on the path between the content provider and consumer and meets any one of the three conditions mentioned above.

To understand how content is cached in the router located along the content delivery path, let us consider the network topology shown in Figure 6.2. When the content Name arrives at router $R_6$, which is requested by Consumer1 and served from the original source, the caching decision at $R_6$ is made in cooperation with the leader $R_5$ by setting the *CachingFlag* to True. If the content is not already present within the community and $R_6$ satisfies any one of the three caching conditions, the content is placed at $R_6$; otherwise, Name is cached at another router within the community that satisfies any of the three conditions. If the content is cached at $R_6$, then no other router in Community1 is eligible to cache the content Name, in order to avoid redundancy within the community.

## 6.5 Performance Evaluation

This section presents the simulation setup, network topologies, and provides an analysis of the results.

### 6.5.1 Simulation Settings

In order to evaluate the performance of our proposed caching methods, PePC and CPePC, here we also use the Icarus simulator [135]. In our simulations, content requests are generated using a Poisson distribution with 10 requests per second, which means

that 10 requests are expected every second on average. Similar to the experiments presented in other chapters, here we also use the Zipf distribution [23] to model content popularity, with the skewness parameter $\alpha$ varied between 0.6 and 1.2.

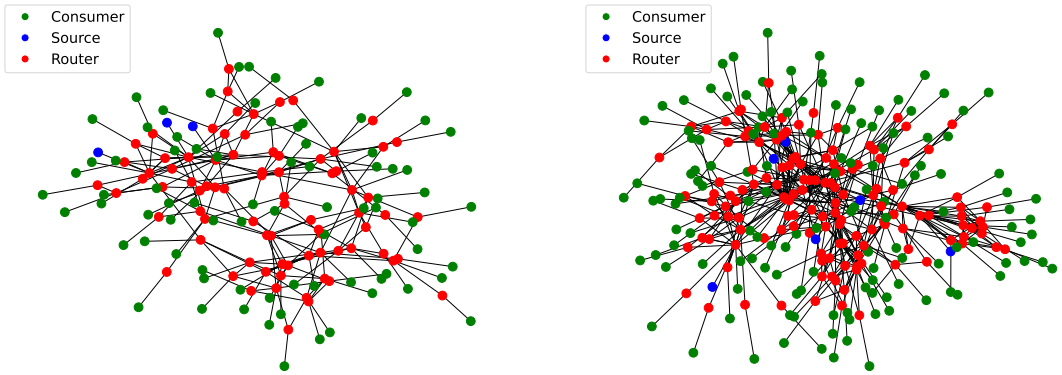
The results were generated using the following simulator settings unless stated otherwise: the simulation content catalog (*i.e.,* total number of unique content objects) contained $10^4$ distinct contents, uniformly distributed among sources. The initial phase involved a warm-up period [50] of $5 \times 10^4$ requests to reach a stable state, which were not included in the performance evaluation. Following the warm-up phase, a total of $10^5$ content requests were generated collectively by all consumers and used for performance evaluation. Assuming that each router in the topology is capable of caching content with the same capacity. The storage capacity of routers varies between 0.05% to 0.25% of the simulation catalog size. In all experiments, Least Recently Used (LRU) [122] serves as the default replacement policy for all caching methods to evict content and accommodate new arrivals, except when evaluating the impact of different replacement policies. In the network, all packets are forwarded towards the upstream and downstream nodes using Dijkstra's weighted shortest path algorithm. The simulation parameters for `PePC` and `CPePC` are configured as follows: the minimum and maximum threshold parameters ($\rho_1$ and $\rho_2$) are set to 0.2 and 0.6, respectively; $P_{max}$ value in Equation 6.8 is set to 1; $\lambda$ is set to 0.125 to prioritize recent data over historical data; weight factor $\omega$ for cache occupancy estimation is set to 0.125, and `T` is set to 10 seconds. Every experiment is run ten times, and the plots are generated using 95% confidence intervals. Table 6.2 summarizes the most important simulation parameters.

### 6.5.2 Network Topology Setup

We evaluate the effectiveness of our proposed `PePC` and `CPePC` techniques through simulations on two RocketFuel ISP topologies [146, 147]: Exodus (AS 3967) with 161 nodes, and AboveNet (AS 6461) with 282 nodes. Figure 6.4 shows the ISP maps of these topologies, where consumer nodes are depicted in green, source nodes in blue, and router nodes in red.

Table 6.2: Simulation Parameter Settings

| **Parameters** | **Value** |
|---|---|
| Network topology | Exodus (AS 3967) and Abovenet (AS 6461) |
| Content catalog size | $10^4$ objects |
| Number of requests | $10^5$ objects |
| Number of warm-up requests | $5 \times 10^4$ objects |
| Content request arrival rate | Poisson distribution, $\lambda = 10$ requests per second |
| Content popularity model | Zipf distribution, $\alpha \in [0.6, 0.8, 1.0, 1.2]$ |
| Routers cache capacity | [0.05 to 0.25]% of catalog size |
| Content placement | Uniform distribution |
| Replacement strategy | LRU, Random, and PLFU |
| Experiment repetitions | 10 |
| Performance metrics | Cache Hit Ratio, Content Access Time, Average Hit Distance, and Cache Diversity |
| Benchmark schemes | LCE, DPCP, MNU, PaCC, and CRUS |




(a) AS 3967 (b) AS 6461

Figure 6.4: RocketFuel ISP Topologies: AS 3967 and AS 6461.

Here, we follow a process similar to that used in Chapter 3 to create the network topologies from the ISP map files. Consumers and sources are artificial nodes with a degree of one, connected to routers with degrees greater than one. The top 5% of routers with the highest degrees are connected to source nodes. We integrated the Python-based Louvain community detection algorithm into Icarus to divide the large

network topologies into several communities. Equation 6.15 is used to determine the number of communities in each topology. We configured the $\tau$ value to 0.15 for both topologies across all simulation results, except in the case of experiments involving varying community sizes, where we explored different community size configurations.

$$N = \lceil \tau \cdot V \rceil \tag{6.15}$$

where $N$ is the total number of communities, $V$ is the total number of nodes in the topology, and $\tau$ represents the scaling factor used to determine the number of communities. Further details regarding both topologies are provided in Table 6.3.

Table 6.3: Details of Network Topologies

| Topology | Nodes | Links | Consumers | Sources | Routers |
|---|---|---|---|---|---|
| Exodus (AS 3967) | 161 | 229 | 79 | 3 | 79 |
| Abovenet (AS 6461) | 282 | 516 | 138 | 6 | 138 |

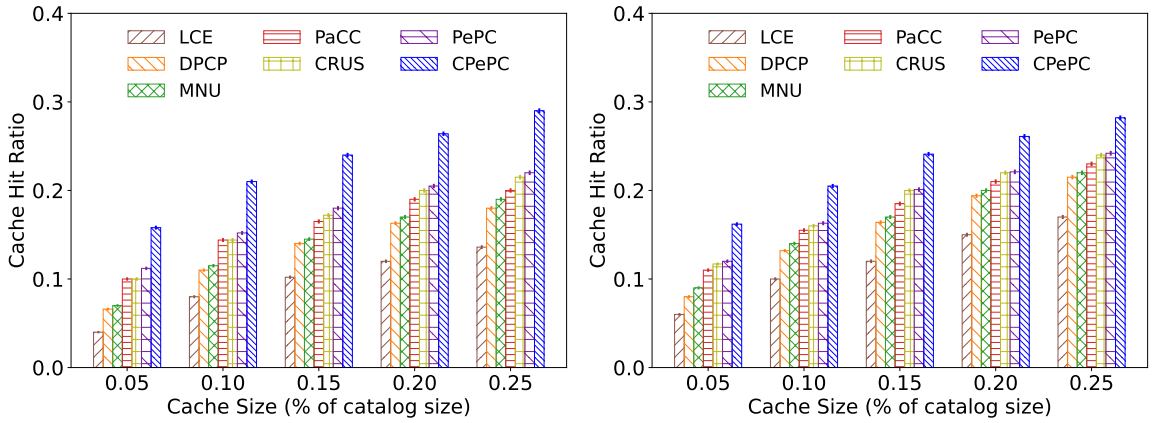
### 6.5.3 Simulation Results and Observations

Here, we present the simulation results for our proposed caching techniques, PePC and CPePC, alongside five benchmark schemes, evaluated under two network topologies and varying simulation parameters. The proposed techniques are compared with benchmark methods (LCE [70], DPCP [176], MNU [75], PaCC [7], and CRUS [93]) using performance metrics such as Cache Hit Ratio, Content Access Time, Average Hit Distance, and Cache Diversity, as used in Chapter 3. Among these benchmark schemes, LCE and MNU are popularity-agnostic techniques, whereas DPCP, PaCC, and CRUS are popularity-based caching techniques.

***Impact of Varying the Cache Sizes:*** We first evaluate the performance of all seven caching techniques by varying the cache size from 0.05% to 0.25% of the catalog size while keeping the Zipf $\alpha$ value fixed at 0.8.

Figures 6.5a and 6.5b show that increasing the cache size increases the cache hit ratio of all seven methods on both topologies. A larger cache size enables routers

to cache more content. When the cache size is relatively small (*i.e.,* 0.05% of the catalog size), we observe that CPePC improves the cache hit ratio by up to 58% on AS 3967 and up to 39% on AS 6461 compared to CRUS. Owing to cooperative content searching and caching, CPePC also achieves improvements of up to 41% on AS 3967 and up to 35% on AS 6461 compared to PePC (second-best method). Furthermore, even with larger cache sizes (*i.e.,* 0.25% of the catalog size), we observe improvements of up to 31.8% on AS 3967 and 16.5% on AS 6461 compared to PePC, and 34.9% on AS 3967 and 17.5% on AS 6461 compared to CRUS. The greater improvement in hit rates is more pronounced for smaller cache sizes because, as cache capacity increases, all strategies tend to accommodate more content within the network routers, thereby reducing performance differences.

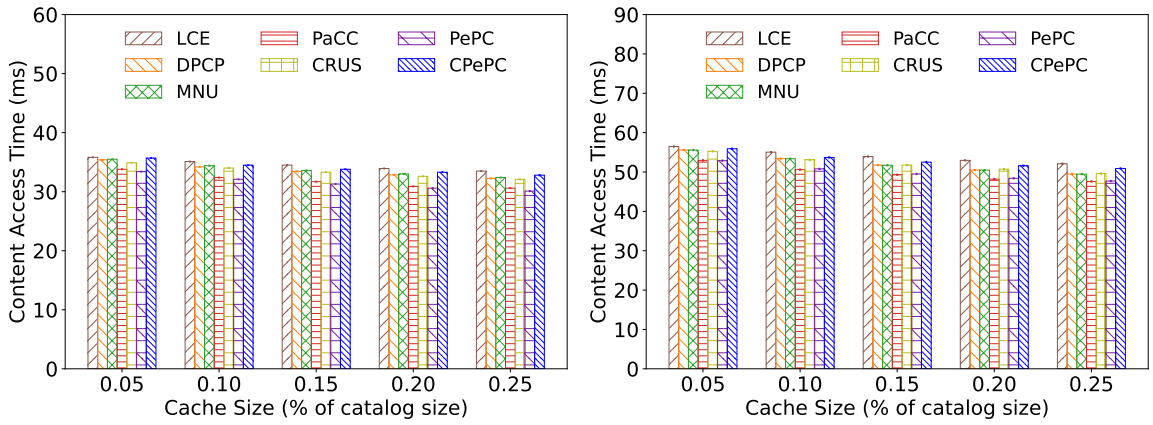



(a) AS 3967 (b) AS 6461

Figure 6.5: Cache Hit Ratio for Different Cache Sizes on Different Network Topologies (Zipf $\alpha = 0.8$).

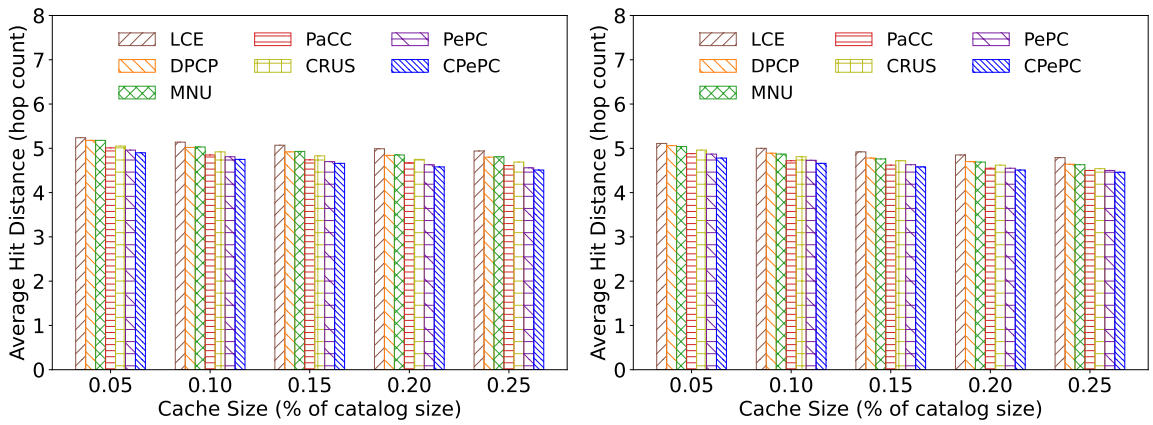
Figure 6.6 illustrates the impact of content access time while increasing the cache size for all seven methods across both topologies. As the cache size increases, content access time for all seven methods decreases because content is served from nearby routers. The content access time of the CPePC strategy is up to 2.1% lower on AS 3967 and up to 2.6% lower on AS 6461 compared to LCE. However, compared to the non-cooperative version PePC, the access time of CPePC increases by up to 8.2% on AS 3967 and up to 6.3% on AS 6461. This increase is due to CPePC relying on the

leader node to search for content within the community, which introduces delays in content retrieval. The extent of this delay depends on the size of the community and the distance between the leader and other routers within it. The effect of delay on community size is discussed later in this section.




(a) AS 3967 (b) AS 6461

Figure 6.6: Impact of Cache Sizes on Content Access Time Across Different Network Topologies (Zipf $\alpha = 0.8$).



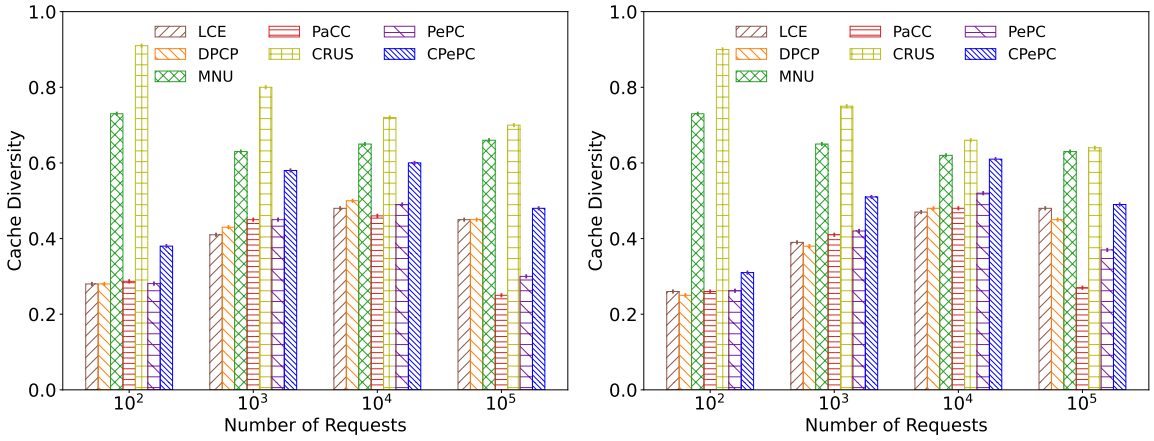

(a) AS 3967 (b) AS 6461

Figure 6.7: Impact of Cache Sizes on Average Hit Distance Across Different Network Topologies (Zipf $\alpha = 0.8$).

Figure 6.7 illustrates the relationship between caching strategies and average hit distance in terms of hop count while varying cache sizes. A lower hit distance means

that the Data packet travels fewer hops between the provider and the consumer. As the cache size increases, the hit distance decreases for all caching methods because content is more readily available in nearby routers. Notably, our PePC and CPePC strategies consistently outperform the other five caching techniques across all cache sizes. On the AS 3967 topology, CPePC achieves hop count reductions of up to 8.7% compared to LCE, 3.8% compared to CRUS, 2.2% compared to PaCC, and 1.27% compared to the non-cooperative version PePC. Similarly, on the AS 6461 topology, CPePC outperforms other strategies with reductions of up to 6.9% compared to LCE, 3.6% compared to CRUS, 2.1% compared to PaCC, and 1.87% compared to PePC. The superior performance of CPePC can be attributed to effective cooperative routing and caching within the community.

***Impact of Varying the Number of Requests:*** Similar to the previous chapters, we measured the cache diversity metric by generating a fixed number of consumer requests: $10^2, 10^3, 10^4$, and $10^5$. This is because the diversity of cached content changes over time as new content is added to the cache and older content is evicted.



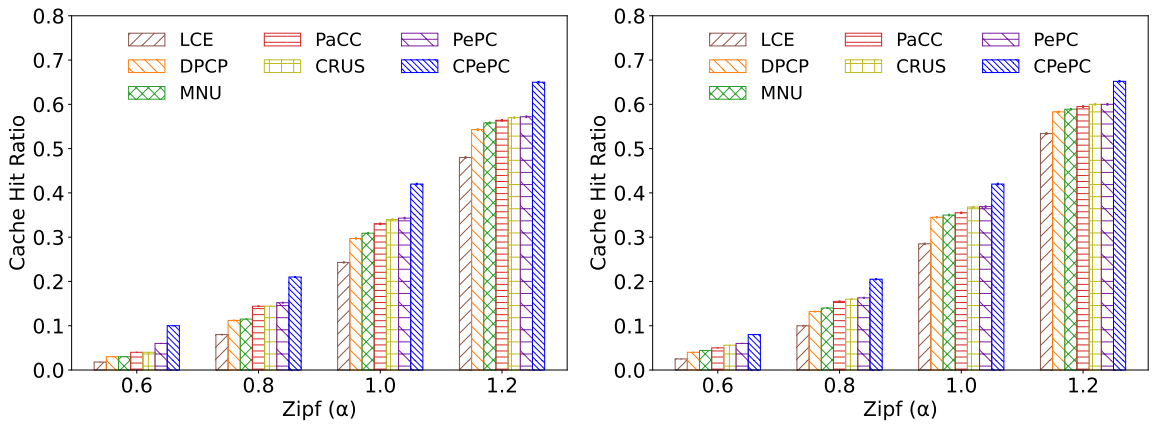

(a) AS 3967 (b) AS 6461

Figure 6.8: Cache Diversity Comparison for Different Network Topologies, with Cache Size = 0.1% and $\alpha = 0.8$.

Figure 6.8 compares cache diversity among all seven caching strategies across both topologies. From Figure 6.8, we can see how the cache diversity for each strategy changes with the number of requests. The diversity of MNU and CRUS strategies

is higher compared to others because these strategies cache content in a few selected routers along the delivery path. While this approach reduces content redundancy, it may lead to poor utilization of the available cache space in the network. Therefore, it is important to balance cache diversity, cache hit ratio, and content access time, since higher diversity does not always result in better hit ratios or faster content access. Our proposed CPePC efficiently utilizes cache space by considering which content to cache and when to cache it, based on the router's occupancy and the popularity of the content. However, CPePC experiences slight redundancy in overall cached content because it ensures no redundancy within communities, although the same content may be cached across multiple communities. Content availability in multiple communities in CPePC increases the cache hit ratio and reduces the load on the content source.

***Impact of Varying Content Popularity Parameter*** $\alpha$***:*** Content popularity may fluctuate significantly over time. To understand how different caching techniques perform under different popularity conditions, we vary the Zipf $\alpha$ (popularity parameter) value from 0.6 to 1.2, which controls the content request patterns. A smaller $\alpha$ corresponds to a situation where a large number of contents have similar request frequencies, while higher $\alpha$ values indicate that a small subset of contents receives a significant number of requests, while the majority receive relatively few.

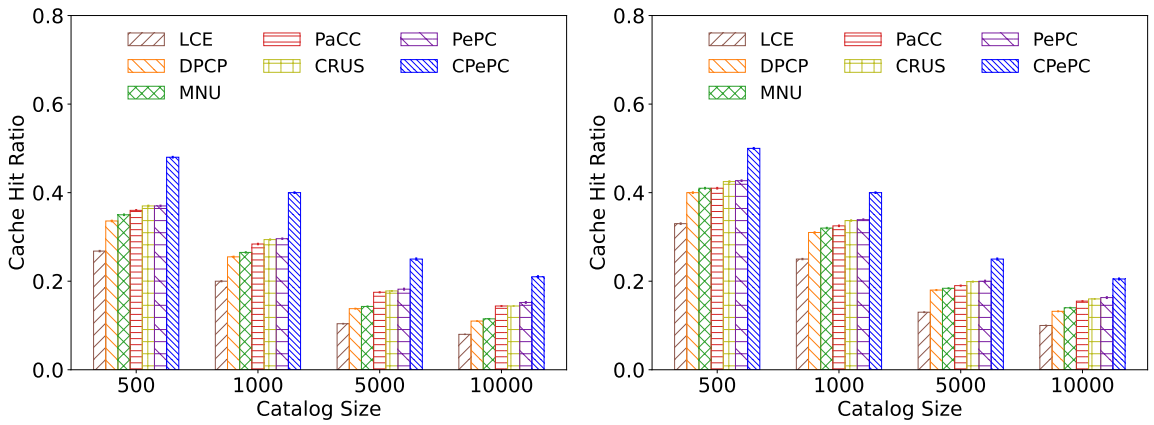



(a) AS 3967 (b) AS 6461

Figure 6.9: Impact of Zipf $\alpha$ on Cache Hit Ratio Across Different Network Topologies (Cache Size = 0.1%).

Figure 6.9 illustrates that as the $\alpha$ value increases, the cache hit ratio of all seven caching techniques increases significantly. This is attributed to a few contents that heavily dominate the request patterns, and these frequently requested contents are present in the router. Notably, our CPePC consistently achieves a higher cache hit ratio on both topologies, regardless of content popularity, compared to the other caching methods. Figures 6.9a and 6.9b illustrate this improvement, showing an increase in cache hit ratio of up to 66.6% on AS 3967 and up to 33.3% on AS 6461 compared to the non-cooperative version, PePC (second-best method). This improvement is due to cooperative searching and caching of content, which enhances cache utilization.

***Impact of Varying Catalog Size:*** Here, we present the simulation results illustrating the impact of catalog size (number of distinct contents) on the cache hit ratio. For this evaluation, each router is allocated a fixed cache size of 10 slots, indicating that each router can store a maximum of 10 contents in its cache (*i.e.,* each content item has the same size), and Zipf $\alpha$ is set to 0.8.




(a) AS 3967 (b) AS 6461

Figure 6.10: Cache Hit Ratio with Varying Catalog Sizes Across Different Network Topologies (Zipf $\alpha = 0.8$).

Figure 6.10 depicts the effect of catalog size on the cache hit ratio for all seven strategies across both topologies. Figures 6.10a and 6.10b show that when the catalog size is small, the cache hit ratio increases for all the caching techniques. However, the cache hit ratio declines when the catalog size is large. This is due to the fact

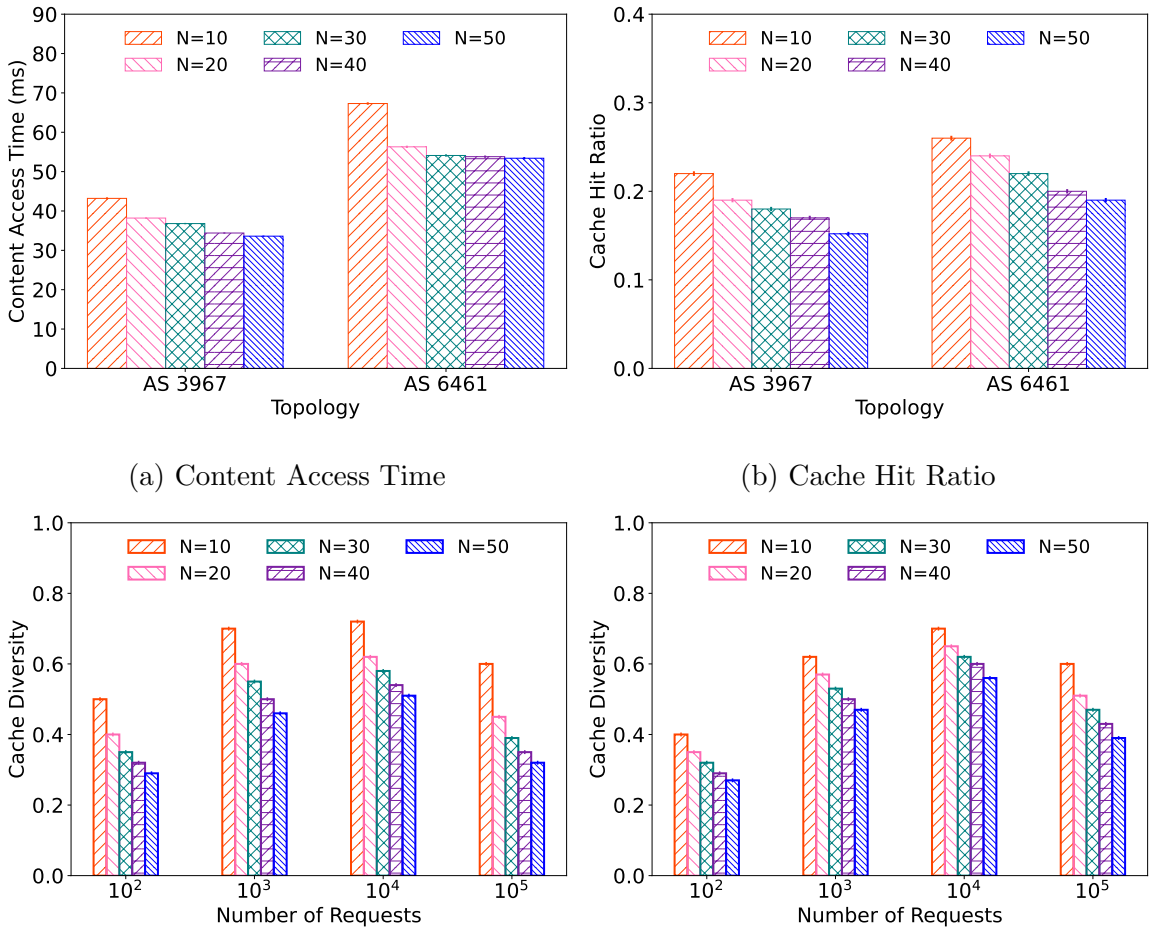
that with a larger catalog size, consumers request a more diverse range of content, making it less likely for all content to be available in the routers due to their limited capacity. In contrast, with a smaller catalog size, a higher proportion of consumer requests can be satisfied from the routers. For example, if the catalog contains only 100 different items, repeated consumer requests for this limited set make them more likely to be in the cache. On the other hand, if the catalog contains 10,000 items, repeated requests for the same content become less frequent, reducing the likelihood of those items being cached. Notably, our CPePC technique outperformed the other caching techniques regardless of catalog size. The higher cache hit ratio of CPePC is achieved through cooperative content searching and caching.

***Impact of Varying Community Sizes:*** CPePC divides the network topology into several communities, and the size of these communities significantly affects its performance. Here, we study the impact of community size on the performance of CPePC.

Figure 6.11 shows the impact of varying community sizes on both topologies with respect to the content access time, cache hit ratio, and cache diversity. We vary the number of communities from $N = 10$ to 50 while keeping fixed Cache Size = 0.1% and Zipf $\alpha = 0.8$. The number of communities can be varied by setting different values to the resolution parameter in the Louvain algorithm [21]. When the resolution value is set below 1, the algorithm generates larger-sized communities, whereas values above 1 generate smaller-sized communities. A larger number of communities means each community has fewer nodes, while a smaller number of communities results in more nodes per community.

Figure 6.11a depicts the content access time of the CPePC across both topologies. From Figure 6.11a, we can see that the content access time decreases with an increasing number of communities. This trend is due to the fact that an increase in the number of communities reduces the size of individual communities, bringing the leader closer to other routers in the community, and thereby reducing content search delays within the community. On the other hand, fewer communities result in larger community sizes, leading to higher content access time when searching for content through the leader.

Figure 6.11b shows that the CPePC achieves a higher cache hit ratio when the number of communities is less. However, as the number of communities increases, the cache hit ratio decreases significantly. The decrease in cache hit ratio with the increasing number of communities is due to the same content being cached at multiple communities. This occurs because the CPePC method ensures zero redundancy within each community, but the same content may be cached across different communities. Hence, when content is transmitted between content providers and consumers, it may traverse a larger number of communities, resulting in content redundancy due to storing the same content in multiple communities and subsequently reducing the cache hit




(a) Content Access Time

(b) Cache Hit Ratio

(c) Cache Diversity (AS 3967)

(d) Cache Diversity (AS 6461)

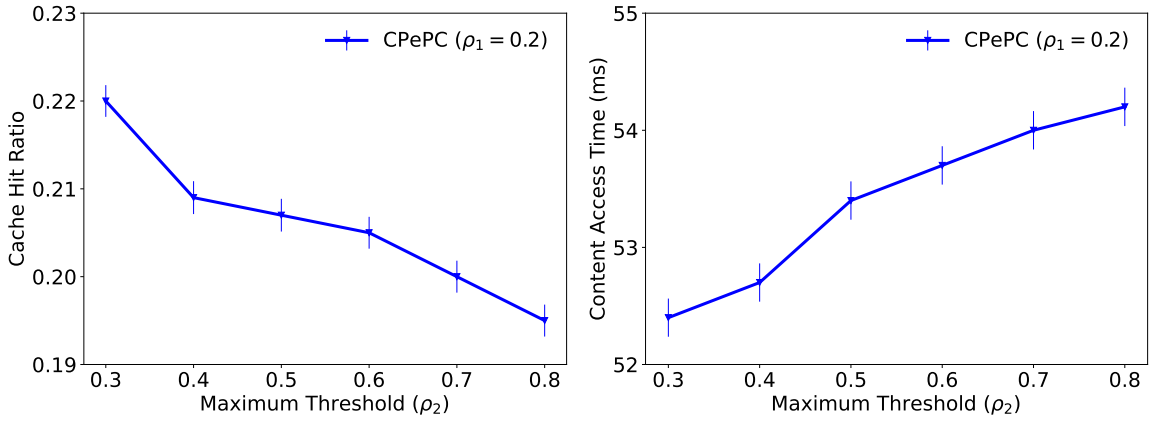
Figure 6.11: CPePC Performance on Different Topologies with Varying Community Sizes (N=10 to N=50) with fixed Cache Size = 0.1% and Zipf $\alpha = 0.8$.

ratio. Choosing an appropriate community size is thus crucial for achieving a balance between cache hit ratio and latency.

Similar to the previous experiments, here too we measured cache diversity after the consumer generated $10^2$, $10^3$, $10^4$, and $10^5$ requests, as diversity changes with the number of requests. Figures 6.11c and 6.11d show the cache diversity of `CPePC` under the impact of different community sizes. When the number of communities is smaller (e.g., $N = 10$), cache diversity is higher because the chances of redundancy are lower. As the number of communities increases (*i.e.,* the size of each community decreases), the diversity of `CPePC` decreases since content redundancy increases. This is because `CPePC` avoids redundancy within a community, but the same content may still be cached in other communities. For example, if $N = 10$, in the worst-case scenario, content `Name` could be cached in all 10 communities, resulting in 10 copies across the network. If $N = 50$, then 50 copies of content `Name` may be cached in the network in the worst-case scenario. So, a smaller number of communities leads to better cache diversity, while a higher number of communities results in lower cache diversity.

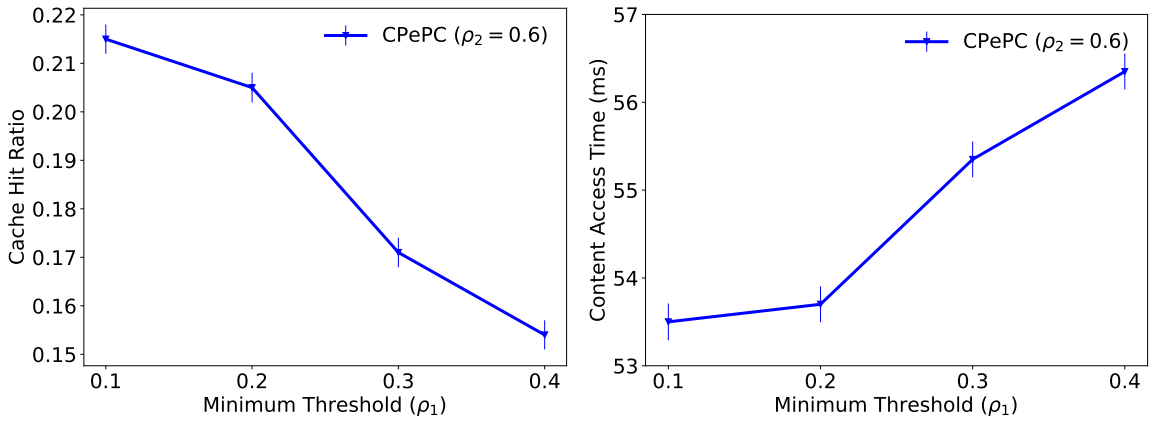
***Impact of Varying Caching Threshold Parameters:*** Here, we explore the impact of adjusting the minimum and maximum caching threshold parameters ($\rho_1$ and $\rho_2$) in Equations 6.4 and 6.5 for `CPePC`. We analyze the effects of varying $\rho_1$ and $\rho_2$ on the cache hit ratio and content access time, while maintaining a fixed cache size of 0.1% and a popularity constant $\alpha$ value of 0.8, using the AS 6461 topology. Fine-tuning the values of $\rho_1$ and $\rho_2$ plays a crucial role in balancing cache hit ratio and content access time. The choice of these values significantly impacts the performance of the caching system.

Figure 6.12 illustrates the cache hit ratio and content access time. This analysis involves varying the parameter $\rho_2$ within the range of 0.3 to 0.8 while keeping $\rho_1$ fixed at a value of 0.2. Figures 6.12a and 6.12b show that as the $\rho_2$ value increases, the cache hit ratio starts to decrease, and latency increases. This behavior can be attributed to the fact that with a lower $\rho_2$ value, the average cache occupancy quickly approaches its maximum threshold ($max_{th}$). Thus, content is only cached in the router when its popularity exceeds the dynamic threshold ($\Delta$). Consequently, content




(a) Cache Hit Ratio (b) Content Access Time

Figure 6.12: Performance Analysis of Cache Hit Ratio and Content Access Time with Varying $\rho_2$.




(a) Cache Hit Ratio (b) Content Access Time

Figure 6.13: Performance Analysis of Cache Hit Ratio and Content Access Time with Varying $\rho_1$.

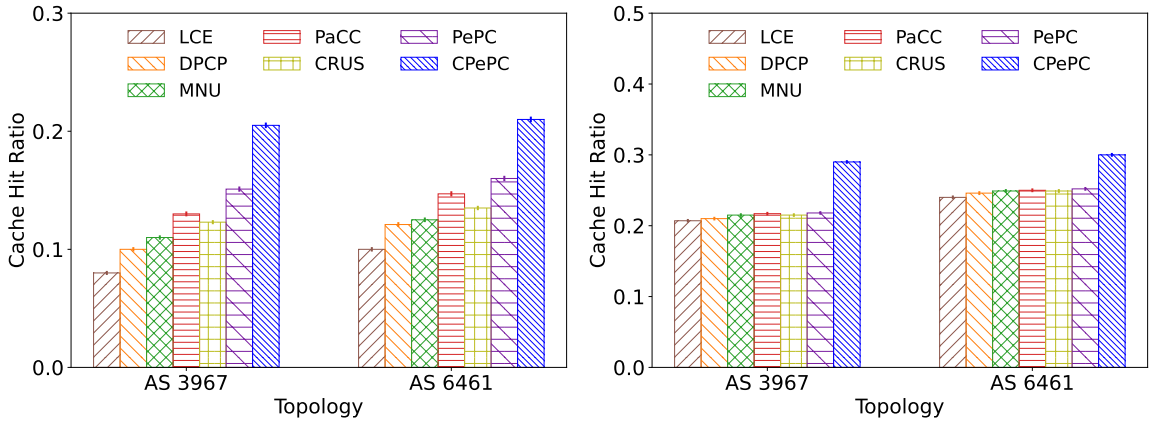
with lower popularity is cached less. Figure 6.13 illustrates the cache hit ratio and content access time metrics, with $\rho_1$ varying between 0.1 and 0.4 while $\rho_2$ is held constant at 0.6. Figures 6.13a and 6.13b show that as the value of $\rho_1$ increases, the cache hit ratio starts to decrease while latency increases. This trend is due to the fact that when $\rho_1$ is set to a lower value, the average cache occupancy quickly reaches its minimum threshold occupancy ($min_{th}$). This results in better cache utilization,

where initially all content is cached, and later, high-popularity content is stored in the router, potentially displacing less popular content.

***Effect of Different Replacement Policies on Caching Techniques:*** So far, all of the aforementioned simulation results have been reported using the LRU policy. Due to the simplicity and effectiveness of LRU, it is widely adopted in the literature as a replacement technique for NDN caching algorithms [7, 50, 93]. LRU performs well in scenarios where content access patterns exhibit high temporal locality. However, its performance degrades when temporal locality is poor, such as in cases where consumer requests are widely distributed. For example, when content requests exhibit little repetition over short intervals, such as in a round-robin request pattern, the cache cannot keep items for a long time, so most requests end up as cache misses. To better understand the caching performance of `PePC` and `CPePC` under different replacement policies, we also explored other replacement algorithms, namely Perfect Least Frequently Used (PLFU) [46] and the Random policy [117]. PLFU tracks access frequencies for all content requests, even if the content is not currently in the cache, and evicts the content with the lowest access frequency, while the Random policy chooses the content to evict at random without regard for access frequency. PLFU requires an additional data structure to maintain the access frequency of the content, making it computationally more expensive compared to LRU and Random. Additionally, search and replacement tasks take more time compared to LRU and Random policies, as the router needs to compare the frequency of the newly arrived content with the content already in the cache.

Here, we analyze and compare the proposed `PePC` and `CPePC` content placement techniques with five other caching techniques across both network topologies under the PLFU and Random replacement policies.

Figure 6.14 shows the cache hit ratios of different caching methods using PLFU and Random replacement policies. Figure 6.14a shows the performance of caching techniques using the Random replacement strategy, in which content is randomly selected from the router to make way for new content when the capacity of the router is full. Figure 6.14b shows the performance analysis using the Perfect-LFU (PLFU),




(a) Random (b) PLFU

Figure 6.14: Cache Hit Ratio Comparison with Different Replacement Policies on Different Network Topologies (Cache Size = 0.1%, Zipf $\alpha = 0.8$).

in which content is evicted based on frequency, with lower-frequency content being removed from the router upon the arrival of new content. The performance of all seven caching techniques improved when using the PLFU eviction policy. However, PLFU requires maintaining the frequency of previously requested items, which adds overhead. From Figure 6.14, we can observe that the proposed caching techniques outperform the other five caching techniques on both topologies, regardless of the replacement policies used for content eviction. Furthermore, `CPePC`, the cooperative version of `PePC`, performs significantly better than `PePC` due to the cooperative management of caches.

### 6.5.4 Analysis of Signaling Overhead

Simulation studies show that `PePC` and `CPePC` outperform several existing NDN caching techniques under various conditions, including different cache sizes, content popularity parameters, catalog sizes, and replacement policies. However, `CPePC` performs routing and caching operations in coordination with the leader node of each community, which introduces additional message exchanges compared to on-path caching techniques. In the following, we examine the overhead associated with implementing

the CPePC technique.

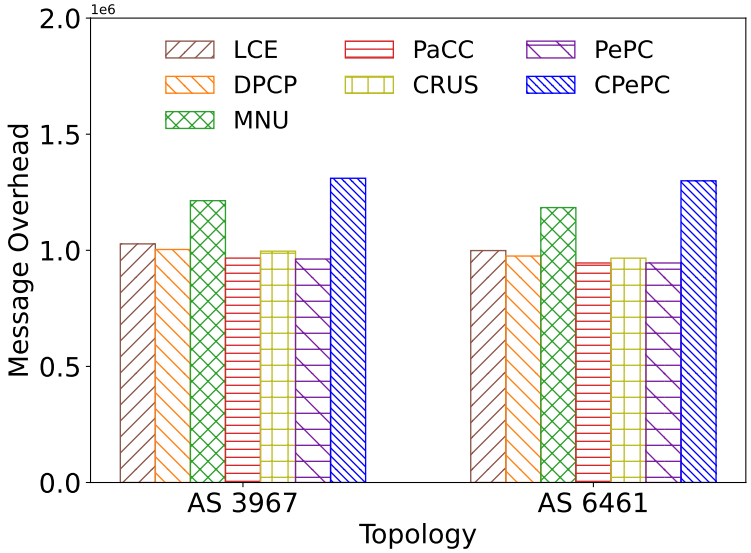



Figure 6.15: Comparison of Total Number of Messages Exchanged Across Different Network Topologies (cache size = 0.1%, Zipf $\alpha = 0.8$).

Figure 6.15 presents a comparison of the number of messages exchanged for routing, caching, and popularity estimation across all seven caching techniques. The number of messages represents the total messages exchanged in the network to retrieve content requested by consumers from the provider. On-path techniques do not require any additional messages other than those necessary to send Interest and Data packets between the consumer and provider. The simulation was conducted on the AS 3967 and AS 6461 topologies, configuring the cache size to 0.1% and Zipf $\alpha$ to 0.8 while keeping the remaining parameters consistent with those outlined in the subsection 6.5.1. Figure 6.15 shows that MNU requires additional messages compared to other on-path techniques (LCE, DPCP, PaCC, CRUS, and PePC) because, in MNU, the content provider coordinates with the network manager to determine the best routers for caching along the content delivery path. On the other hand, the CPePC includes additional messages that are required for coordination with the community leader nodes. These additional messages include content searching within the community through the leader node while forwarding Interest packets, coordinating with the leader node during content caching, and leader nodes exchanging popularity informa-

tion with other leader nodes of the communities in the network.

## 6.6 Discussion

In PePC, each router individually performs content searching, caching, and popularity estimation, so it does not introduce communication overhead. PePC only incurs storage overhead, as each router maintains a table for content popularity estimation. However, because content searching decisions are made without cooperation, PePC cannot utilize content available outside the request forwarding path. In addition, its independent caching decisions can increase content redundancy, as multiple routers along the content delivery path may end up caching the same content. Unlike PePC, CPePC enhances content distribution and cache utilization through a community division approach, which helps overcome the limitations of on-path caching strategies. CPePC also addresses the challenges of operating the network using a centralized controller [186] by dividing the large network into smaller groups and selecting a leader within each group. However, these advantages come with increased costs of content searching and caching.

*Storage Overhead:* PePC requires more storage for popularity estimation, as each router maintains a table to track the frequency of each content received. In a topology with $N$ routers and each table of size $s$ bytes, the total storage required is $N \times s$ bytes. For example, if $N = 10$ and $s = 100$ bytes, the total storage is 1000 bytes. However, in CPePC, only the leader node of each community maintains the popularity table. If the network is divided into $M$ communities and each leader node's table is $s$ bytes, the total storage is reduced from $N \times s$ bytes to $M \times s$ bytes. For instance, if $M = 2$ and $s = 100$ bytes, the total storage is only 200 bytes.

*Design Complexity:* CPePC extends the NDN packet format specification [3] by adding a few additional fields to enable more efficient content searching and caching within a community. Using this modified packet structure reduces overhead and communication latency between routers and the community leader. However, extending the native NDN packet structure requires support from routers, which may introduce design

complexity.

*Scalability:* The community division approach of CPePC alleviates the load on a single centralized node (controller) for managing the network and also reduces the burden on other routers for routing, caching decisions, and popularity estimation. However, having a single leader within each community can result in a potential single point of failure and may cause a bottleneck at the leader node as the community grows. This issue can be addressed by extending the approach and having backup leader nodes within the community. Another issue that warrants deployment considerations is the management of the community structure and its regular updates, as nodes may fail or leave the community.

*Content Search Cost:* In CPePC, content is searched within a community through a leader node. This simplifies the search process but introduces a delay in content retrieval. The content search cost depends on how far the leader node is from the requester. An increase in community size can lead to higher content searching costs. So, it is important to determine the optimal community size to balance the content search cost.

## 6.7 Summary

In this chapter, we described PePC and CPePC, predictive popularity-based caching techniques for NDN. PePC and CPePC make caching decisions by dynamically estimating cache occupancy and content popularity. CPePC extends PePC by dividing the network topology into multiple communities and designating a leader node in each community to facilitate cooperation among the routers. These leader nodes help reduce the load on individual routers by taking responsibility for content searching, caching, and both local and global content popularity estimation. The community division also improves cache diversity by ensuring that only a single copy of each content is cached within the community. We compared the performance of PePC and CPePC with well-known caching techniques from the NDN literature, such as LCE, DPCP, MNU, PaCC, and CRUS, through simulation studies on ISP topology graphs. The

results demonstrate that making caching decisions based on both the availability of cache space at routers and the popularity of content improves the cache hit ratio and response time while reducing content redundancy.

# Chapter 7

# Conclusion and Future Work

In this chapter, we summarize the contributions of this thesis and discuss potential future directions to extend our work.

## 7.1 Conclusion

The growing demand for multimedia content has led to the design of a new Internet architecture that addresses the limitations of existing architectures in content delivery. Caching content within network routers and content distribution over multiple links makes Named Data Networking (NDN) a promising Internet architecture for the future. However, each router can be equipped with a limited cache capacity. A larger cache size also results in longer access time. This limitation in storage poses a significant challenge in efficiently caching content to meet the varied demands of users. In this thesis, we have designed several cooperative caching techniques for NDN aimed at better utilizing the limited storage capacity of routers. The major purpose of our studies is to enhance content discovery and cache more relevant content in network routers. We addressed the performance limitations of the on-path approach for request forwarding by exploring content available in both on-path and off-path routers. We evaluated the performance of our proposed cooperative caching techniques by implementing them in the popularly used Icarus simulator. Simulations were conducted on realistic ISP network topologies under various conditions, including changes in cache

size, request patterns, number of requests, content size, catalog size, and replacement policies.

The specific contributions of our thesis are summarized below.

***Contribution 1:*** In Chapter 3, we described `eNCache`, a cooperative content searching and caching technique. `eNCache` collaboratively searches for content in neighboring routers while forwarding the request towards the provider. On the return path, routers also cache the content in cooperation with their neighbors. The major benefit of `eNCache` is that it can fetch content from the nearest on-path or off-path routers if the content is available, without incurring additional delay. To achieve off-path forwarding, we modified the Interest packet structure of NDN. We use separate packets for on-path and off-path forwarding. In the on-path Interest packet, we added a field called *VisitedNodes*, which carries information about the nodes visited by the previous hop. This allows the next router to forward the request to off-path routers that are not part of the *VisitedNodes* list. *VisitedNodes* field helps reduce the number of request messages needed to be sent to off-path neighbors. In `eNCache`, on-path and off-path forwarding are performed simultaneously. So even if the content is missed at off-path routers, it does not add any additional delay, as the request is already being forwarded towards the original content source. We conducted simulations on large-scale network topologies using various performance metrics to compare `eNCache` with state-of-the-art on-path and off-path caching techniques. The simulation results show that, under various conditions, `eNCache` outperformed other on-path and off-path techniques.

***Contribution 2:*** Cooperative decisions for content lookup and caching improve performance. However, due to the limited capacity of routers, caching all content received at a router increases the replacement rate, as new arrivals require space when the cache is full. The frequent replacement can negatively impact caching efficiency. Therefore, it is better to distinguish between popular and unpopular content when making caching decisions, so that popular content remains in the cache for a longer time. To implement this idea, we introduced `PeNCache` in Chapter 4, a popularity-based cooperative content lookup and caching technique. In `PeNCache`, each router maintains a popularity table to track passing requests. In addition to local popularity, `PeNCache`

also considers the global popularity of content. For global popularity estimation, we identify a few nodes with high degree centrality in the network as leaders, which help in determining the global popularity of content. Simulation outcomes demonstrate that `PeNCache` outperformed both popularity-agnostic and popularity-based cooperative and non-cooperative caching techniques across various realistic network topologies and performance metrics.

***Contribution 3:*** In large-scale network topologies, establishing cooperation becomes challenging. To address this, Chapter 5 designed `DPCCache`, which divides the topology into multiple communities or clusters to easily establish collaboration among nodes. `DPCCache` uses a community detection algorithm to form multiple non-overlapping communities. In each community, a router with the highest degree centrality is elected as the leader node. The leader is responsible for searching content, placing content, replacing content, and popularity estimation. The caching decision through the leader node within the community reduces content redundancy, as `DPCCache` ensures that the same content is not cached at more than one place. For popularity estimation, `DPCCache` considers both local (*i.e.,* content popular within the community) and global (*i.e.,* content popular across all communities) popularity of content. `DPCCache` also caches evicted content at another router within the same community to improve the cache hit ratio and reduce overall network traffic. We evaluated the performance of `DPCCache` against state-of-the-art caching techniques under different simulation parameters. The outcomes show that the community-division approach improves content availability in the network, which significantly reduces the load on the original content source.

***Contribution 4:*** Our previous caching techniques (`PeNCache` and `DPCCache`) cached all content as long as cache space was available to better utilize the cache. Once the cache is full, only popular content is cached by replacing less popular content. However, it may be more effective to cache popular content before the cache reaches its full capacity. To implement this idea, Chapter 6 introduced `PePC` and an enhanced version, `CPePC`, predictive popularity-based caching techniques. While making caching decisions, `PePC` and `CPePC` dynamically monitor cache occupancy, enabling more efficient

and adaptive content placement. We set minimum and maximum caching thresholds to manage cache usage. If the occupancy level is below the minimum threshold, all contents are cached as sufficient space is available. If the occupancy reaches the maximum threshold, only popular contents are cached. When the occupancy lies between these two thresholds, a predictive model is used to decide which content to allow into the cache and which to reject. In `PePC`, decisions are made individually for each router. Here, each router performs content searching, caching, and popularity estimation independently. An extension version of `PePC` is presented subsequently, named `CPePC`, where networks are partitioned into communities, and a leader node is elected within each community to facilitate cooperative request routing, content caching, and popularity estimation. Both `PePC` and `CPePC` were compared with several state-of-the-art caching techniques over large-scale network topologies by tuning various simulation parameters. The results show that dynamically estimating cache occupancy and content popularity improves content availability in the network.

## 7.2 Future Work

The performance of our cooperative techniques shows a promising direction for NDN in improving content delivery efficiency and cache utilization. However, our work can be further extended to improve data retrieval and minimize the cooperation overhead. We outline some future directions that could extend the work presented in this thesis.

1. **Real-World Deployment and Evaluation:** In this thesis, we conducted simulation studies of the proposed caching techniques for performance evaluation due to the unavailability of large-scale native NDN testbeds. Future work includes implementing the proposed cooperative techniques on a real-world NDN testbed for evaluation and assessment. The implementation should account for practical aspects such as router cache sizes, diverse content types and sizes, and real Internet traffic patterns. Considering these factors will enable a more accurate assessment of practical performance (e.g., response time and cache hit

ratio) and a more precise analysis of the associated computational resource requirements.

2. **Optimizing Neighbor Discovery:** In our decentralized cooperative techniques, `eNCache` and `PeNCache`, we added a field to the NDN Interest packet to track the neighbors visited by each on-path router. This helps avoid sending messages to routers that have already been explored. However, if the on-path routers do not share common neighbors, this neighbor discovery approach increases communication overhead and the size of the Interest packet. To address this, future work should explore alternative approaches for neighbor discovery that are more efficient in scenarios where on-path routers do not share common neighbors. One approach is to forward packets only to neighbors with higher degree centrality, as they are more likely to have the requested content. Another method involves resetting the visited neighbor field after 2 or 3 hops to avoid increasing the size of the Interest packet. A third solution is to use a Software-Defined Networking (SDN) controller to handle neighbor discovery.

3. **Investigating Dynamic Community Sizes:** We divided the topology into multiple communities to simplify cooperation among nodes in our works, namely `DPCCache` and `CPePC`. The size of these communities and their aggregate cache space remains fixed throughout the simulation. However, it is interesting to explore the shift from fixed-size communities to dynamic ones. Future work involves periodically adjusting the cache capacity of communities based on the number of requests they receive. Cache space can be moved from communities with fewer requests to those experiencing higher requests. For example, if some communities remain idle for a long time with no updates while others receive more requests, allocating additional cache resources to the busier communities can improve performance by allowing them to cache more content for longer periods. However, this requires mechanisms for sharing cache among the communities and keeping track of what portion of the cache is shared with which community, along with their performance implications.

4. **Load Balancing with Multi-Leader Approach:** Our community-centric approach designates a leader node in each community to manage routing, caching, and popularity estimation decisions. Our approaches simplify the management of cooperative decisions and improve content availability, but they are vulnerable to leader node failures. We need mechanisms that can handle such failures. One approach could be to have backup leader nodes or multiple leader nodes within a community. A backup/multi-leader approach can better distribute the workload and also handle more realistic scenarios where, if one leader fails, its responsibilities can be taken over by another leader within the same community.

# Bibliography


[1] Cisco Annual Internet Report. Online. Available: https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/index.html. Last accessed on May 26, 2026.

[2] DataReportal: Global Overview Report 2025. Online. Available: https://datareportal.com/reports/digital-2025-global-overview-report. Last accessed on May 26, 2026.

[3] NDN packet format specification. Online. Available: https://docs.named-data.net/NDN-packet-spec/current/index.html. Last accessed on May 26, 2026.

[4] Alduayji, S., Belghith, A., Gazdar, A., and Al-Ahmadi, S. PF-ClusterCache: popularity and freshness-aware collaborative cache clustering for named data networking of things. *Applied Sciences 12*, 13 (2022), 6706.

[5] Alduayji, S., Belghith, A., Gazdar, A., and Al-Ahmadi, S. PF-EdgeCache: popularity and freshness aware edge caching scheme for ndn/iot networks. *Pervasive and Mobile Computing 91* (2023), 101782.

[6] Alhowaidi, M., Nadig, D., Hu, B., Ramamurthy, B., and Bockelman, B. Cache management for large data transfers and multipath forwarding strategies in named data networking. *Computer Networks 199* (2021), 108437.

[7] Amadeo, M., Campolo, C., Ruggeri, G., and Molinaro, A. Beyond edge caching: Freshness and popularity aware iot data caching via ndn at

internet-scale. *IEEE Transactions on Green Communications and Networking 6*, 1 (2022), 352–364.

[8] Amadeo, M., Campolo, C., Ruggeri, G., and Molinaro, A. Popularity-aware closeness based caching in ndn edge networks. *Sensors 22*, 9 (2022), 3460.

[9] Amadeo, M., Ruggeri, G., Campolo, C., and Molinaro, A. Diversity-improved caching of popular transient contents in vehicular named data networking. *Computer Networks 184* (2021), 107625.

[10] Amadeo, M., Ruggeri, G., Campolo, C., Molinaro, A., and Mangiullo, G. Caching popular and fresh iot contents at the edge via named data networking. In *Proc. IEEE INFOCOM WKSHPS* (2020), pp. 610–615.

[11] Androutsellis-Theotokis, S., and Spinellis, D. A survey of peer-to-peer content distribution technologies. *ACM computing surveys (CSUR) 36*, 4 (2004), 335–371.

[12] Arianfar, S., Nikander, P., and Ott, J. Packet-level caching for information-centric networking. In *Proc. ACM SIGCOMM ReArch Workshop* (2010), vol. 4.

[13] Ascigil, O., Sourlas, V., Psaras, I., and Pavlou, G. Opportunistic off-path content discovery in information-centric networks. In *Proc. IEEE LANMAN* (2016), pp. 1–7.

[14] Badshah, J., Kamran, M., Shah, N., and Abid, S. A. An improved method to deploy cache servers in software defined network-based information centric networking for big data. *Journal of Grid Computing 17* (2019), 255–277.

[15] Ballani, H., and Francis, P. Towards a global ip anycast service. *ACM SIGCOMM Computer Communication Review 35*, 4 (2005), 301–312.

[16] Barish, G., and Obraczka, K. World wide web caching: Trends and techniques. *IEEE Communications magazine 38*, 5 (2000), 178–184.

[17] Bayhan, S., Wang, L., Ott, J., Kangasharju, J., Sathiaseelan, A., and Crowcroft, J. On content indexing for off-path caching in information-centric networks. In *Proc. of the 3rd ACM ICN* (2016), pp. 102–111.

[18] Berger, D. S., Beckmann, N., and Harchol-Balter, M. Practical bounds on optimal caching with variable object sizes. *Proc. ACM on Measurement and Analysis of Computing Systems 2*, 2 (2018), 1–38.

[19] Bernardini, C., Silverston, T., and Festor, O. MPC: popularity-based caching strategy for content centric networks. In *Proc. IEEE Int. Conf. Commun. (ICC)* (2013), pp. 3619–3623.

[20] Blaze, M., and Alonso, R. Dynamic hierarchical caching in large-scale distributed file systems. In *Proc. of the 12th ICDCS* (1992), pp. 521–528.

[21] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment 2008*, 10 (2008), P10008.

[22] Borst, S., Gupta, V., and Walid, A. Distributed caching algorithms for content distribution networks. In *Proc. IEEE INFOCOM* (2010), pp. 1–9.

[23] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. Web caching and zipf-like distributions: Evidence and implications. In *Proc. IEEE INFOCOM'99* (1999), vol. 1, pp. 126–134.

[24] Bu, Z., Li, H.-J., Cao, J., Wang, Z., and Gao, G. Dynamic cluster formation game for attributed graph clustering. *IEEE transactions on cybernetics 49*, 1 (2017), 328–341.

[25] Cao, P., and Irani, S. Cost-aware www proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems (USITS 97)* (1997).

[26] Cerf, V., and Kahn, R. A protocol for packet network intercommunication. *IEEE Transactions on communications 22*, 5 (1974), 637–648.

[27] Chai, W. K., He, D., Psaras, I., and Pavlou, G. Cache "less for more" in information-centric networks (extended version). *Computer Communications 36*, 7 (2013), 758–770.

[28] Chankhunthod, A., Danzig, P. B., Neerdaels, C., Schwartz, M. F., and Worrell, K. J. A hierarchical internet object cache. In *USENIX Annual Technical Conference* (1996), pp. 153–164.

[29] Chaudhary, P., Hubballi, N., and Kulkarni, S. G. NCache: neighborhood cooperative caching in named data networking. In *Proc. 5th HotICN'22* (2022), pp. 36–41.

[30] Chaudhary, P., Hubballi, N., and Kulkarni, S. G. eNCache: improving content delivery with cooperative caching in named data networking. *Computer Networks* (2023), 110104.

[31] Chen, C., Jiang, J., Fu, R., Chen, L., Li, C., and Wan, S. An intelligent caching strategy considering time-space characteristics in vehicular named data networks. *IEEE Transactions on Intelligent Transportation Systems 23*, 10 (2021), 19655–19667.

[32] Chiocchetti, R., Perino, D., Carofiglio, G., Rossi, D., and Rossini, G. INFORM: a dynamic interest forwarding mechanism for information centric networking. In *Proc. 3rd ACM SIGCOMM Workshop Inf.-Centric Netw. (ICN)* (2013), pp. 9–14.

[33] Chiocchetti, R., Rossi, D., Rossini, G., Carofiglio, G., and Perino, D. Exploit the known or explore the unknown? hamlet-like doubts in icn. In *Proceedings of the second edition of the ICN workshop on Information-centric networking* (2012), pp. 7–12.

[34] Cho, K., Lee, M., Park, K., Kwon, T. T., Choi, Y., and Pack, S. WAVE: popularity-based and collaborative in-network caching for content-oriented networks. In *Proc. IEEE INFOCOM Workshops* (2012), pp. 316–321.

[35] Choi, H.-g., Yoo, J., Chung, T., Choi, N., Kwon, T., and Choi, Y. CoRC: coordinated routing and caching for named data networking. In *Proc. 10th ACM/IEEE Symp. Architect. Netw. Commun. Syst. (ANCS)* (2014), pp. 161–172.

[36] Cooper, I., Melve, I., and Tomlinson, G. Internet web replication and caching taxonomy. RFC 3040, 2001.

[37] Dannewitz, C., Kutscher, D., Ohlman, B., Farrell, S., Ahlgren, B., and Karl, H. Network of information (netinf)–an information-centric networking architecture. *Computer Communications 36*, 7 (2013), 721–735.

[38] Danzig, P. B., Hall, R. S., and Schwartz, M. F. A case for caching file objects inside internetworks. *ACM SIGCOMM Computer Communication Review 23*, 4 (1993), 239–248.

[39] Datta, A., Dutta, K., Thomas, H., VanderMeer, D., and Ramamritham, K. Proxy-based acceleration of dynamically generated content on the world wide web: An approach and implementation. *ACM Transactions on Database Systems (TODS) 29*, 2 (2004), 403–443.

[40] Davison, B. D. A web caching primer. *IEEE internet computing 5*, 4 (2001), 38–45.

[41] Deering, S. E. Host extensions for ip multicasting. RFC 1112, 1988.

[42] Detti, A., Blefari Melazzi, N., Salsano, S., and Pomposini, M. CONET: a content centric inter-networking architecture. In *Proc. ACM SIGCOMM Workshop Inf. Centric Netw* (2011), pp. 50–55.

[43] Dhara, S., Majidi, A., and Clarke, S. Revving up vndn: Efficient caching and forwarding by expanding content popularity perspective and mobility. *Computer Communications 212* (2023), 342–352.

[44] Din, I. U., Hassan, S., Khan, M. K., Guizani, M., Ghazali, O., and Habbal, A. Caching in information-centric networking: Strategies, challenges, and future research directions. *IEEE Communications Surveys & Tutorials 20*, 2 (2018), 1443–1474.

[45] Dräxler, M., and Karl, H. Efficiency of on-path and off-path caching strategies in information centric networks. In *Proc. IEEE Int. Conf. Green Comput. Commun. (GreenCom)* (2012), pp. 581–587.

[46] Einziger, G., Friedman, R., and Manes, B. TinyLFU: a highly efficient cache admission policy. *ACM Transactions on Storage (ToS) 13*, 4 (2017), 1–31.

[47] Fan, C., Shannigrahi, S., Papadopoulos, C., and Partridge, C. Discovering in-network caching policies in ndn networks from a measurement perspective. In *Proc. of 7th ACM ICN'20* (2020), pp. 106–116.

[48] Floyd, S., and Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking 1*, 4 (1993), 397–413.

[49] Fotiou, N., Nikander, P., Trossen, D., and Polyzos, G. C. Developing information networking further: From psirp to pursuit. In *Proc. Conf. Broadband Commun. Netw. Syst.* (2012), pp. 1–13.

[50] Gui, Y., and Chen, Y. A cache placement strategy based on compound popularity in named data networking. *IEEE Access 8* (2020), 196002–196012.

[51] Gupta, D., Rani, S., Ahmed, S. H., Garg, S., Piran, M. J., and Alrashoud, M. ICN-based enhanced cooperative caching for multimedia streaming in resource constrained vehicular environment. *IEEE Transactions on Intelligent Transportation Systems 22*, 7 (2021), 4588–4600.

[52] Hail, M. A., Amadeo, M., Molinaro, A., and Fischer, S. Caching in named data networking for the wireless internet of things. In *2015 international conference on recent advances in internet of things (RIoT)* (2015), pp. 1–6.

[53] He, X., Liu, H., Li, W., Valera, A., and Seah, W. K. EABC: energy-aware centrality-based caching for named data networking in the iot. In *Proc. IEEE 25th Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)* (2024), pp. 259–268.

[54] Hou, J., Tao, T., Lu, H., and Nayak, A. Intelligent caching with graph neural network-based deep reinforcement learning on sdn-based icn. *Future Internet 15*, 8 (2023), 251.

[55] Hou, J., Tao, T., Lu, H., and Nayak, A. An optimized gnn-based caching scheme for sdn-based information-centric networks. In *Proc. IEEE GLOBECOM 2023* (2023), pp. 401–406.

[56] Hou, J., Xia, H., Lu, H., and Nayak, A. A gnn-based approach to optimize cache hit ratio in ndn networks. In *Proc. IEEE GLOBECOM* (2021), pp. 1–6.

[57] Hou, J., Xia, H., Lu, H., and Nayak, A. A graph neural network approach for caching performance optimization in ndn networks. *IEEE Access 10* (2022), 112657–112668.

[58] Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., and West, M. J. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS) 6*, 1 (1988), 51–81.

[59] Hu, X., and Gong, J. Opportunistic on-path caching for named data networking. *IEICE Transactions on Communications 97*, 11 (2014), 2360–2367.

[60] Hu, X., Papadopoulos, C., Gong, J., and Massey, D. Not so cooperative caching in named data networking. In *2013 IEEE Global Communications Conference (GLOBECOM)* (2013), pp. 2263–2268.

[61] Hu, X., Yin, J., Zheng, S., Li, R., Cheng, G., and Gong, J. A demand and responsiveness-based caching strategy for network coding enabled ndn. In *Proc. IEEE GLOBECOM* (2020), pp. 1–6.

[62] Hu, X., Zheng, S., Zhang, G., Zhao, L., Cheng, G., Gong, J., and Li, R. An on-demand off-path cache exploration based multipath forwarding strategy. *Computer Networks 166* (2020), 107032.

[63] Huang, W., Song, T., Yang, Y., and Zhang, Y. Cluster-based cooperative caching with mobility prediction in vehicular named data networking. *IEEE Access 7* (2019), 23442–23458.

[64] Ioannou, A., and Weber, S. Towards on-path caching alternatives in information-centric networks. In *Proc. 39th IEEE Conf. Local Comput. Netw. (LCN)* (2014), pp. 362–365.

[65] Ioannou, A., and Weber, S. Exploring content popularity in information-centric networks. *China Communications 12*, 7 (2015), 13–22.

[66] Iqbal, S. M. A., et al. Cache-MAB: a reinforcement learning-based hybrid caching scheme in named data networks. *Future Generation Computer Systems 147* (2023), 163–178.

[67] Iqbal, S. M. A., et al. Cache-MCDM: a hybrid caching scheme in mobile named data networks based on multi-criteria decision making. *Future Generation Computer Systems 154* (2024), 344–358.

[68] Ishigaki, G., Gour, R., Yousefpour, A., Shinomiya, N., and Jue, J. P. Cluster leader election problem for distributed controller placement in sdn. In *Proc. IEEE GLOBECOM* (2017), pp. 1–6.

[69] Jaber, G., and Kacimi, R. A collaborative caching strategy for content-centric enabled wireless sensor networks. *Computer Communications 159* (2020), 60–70.

[70] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. Networking named content. In *Proc. of ACM CoNEXT* (2009), pp. 1–12.

[71] Jin, C., Yao, H., Cai, R., Mai, T., Xu, J., Xiong, Z., and Niyato, D. SkyNDN incentivizer: Enhancing content sharing in uav named data networking. *IEEE Transactions on Mobile Computing* (2025), 1–18.

[72] Jmal, R., and Fourati, L. C. An openflow architecture for managing content-centric-network (OFAM-CCN) based on popularity caching strategy. *Computer Standards & Interfaces 51* (2017), 22–29.

[73] Jung, J., Sit, E., Balakrishnan, H., and Morris, R. Dns performance and the effectiveness of caching. In *Proc. of the 1st ACM SIGCOMM Workshop on Internet Measurement* (2001), pp. 153–167.

[74] Khan, J. A., Westphal, C., and Ghamri-Doudane, Y. A popularity-aware centrality metric for content placement in information centric networks. In *2018 International Conference on Computing, Networking and Communications (ICNC)* (2018), pp. 554–560.

[75] Khandaker, F., Li, W., Oteafy, S., and Hassanein, H. Maximizing producer-driven cache valuation in information-centric networks. In *Proc. IEEE GLOBECOM* (2021), pp. 1–6.

[76] Kim, D., and Kim, Y. Enhancing ndn feasibility via dedicated routing and caching. *Computer networks 126* (2017), 218–228.

[77] Koide, M., Matsumoto, N., and Matsuzawa, T. Caching method for information-centric ad hoc networks based on content popularity and node centrality. *Electronics 13*, 12 (2024), 2416.

[78] Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S., and Stoica, I. A data-oriented (and beyond) network architecture. In *Proc. ACM SIGCOMM* (2007), pp. 181–192.

[79] Korupolu, M. R., and Dahlin, M. Coordinated placement and replacement for large-scale distributed caches. *IEEE Trans. Knowl. Data Eng. 14*, 6 (2002), 1317–1329.

[80] Krishna, P., Vaidya, N. H., Chatterjee, M., and Pradhan, D. K. A cluster-based approach for routing in dynamic networks. *ACM SIGCOMM computer communication review 27*, 2 (1997), 49–64.

[81] Krishnamurthy, B., Wills, C., and Zhang, Y. On the use and performance of content distribution networks. In *Proc. 1st ACM SIGCOMM Workshop on Internet Measurement* (2001), pp. 169–182.

[82] Lai, F., Qiu, F., Bian, W., Cui, Y., and Yeh, E. Scaled vip algorithms for joint dynamic forwarding and caching in named data networks. In *Proc. of 3rd ACM ICN'16* (2016), pp. 160–165.

[83] Laoutaris, N., Che, H., and Stavrakakis, I. The LCD interconnection of lru caches and its analysis. *Performance Evaluation 63*, 7 (2006), 609–634.

[84] Laoutaris, N., Syntila, S., and Stavrakakis, I. Meta algorithms for hierarchical web caches. In *Proc. IEEE Int. Conf. Perform. Comput. Commun. (IPCCC)* (2004), pp. 445–452.

[85] Lee, J., and Kim, D. Heartbeat: effective access to off-path cached content in ndn. *IEEE Transactions on Network Science and Engineering 12*, 4 (2025), 2974–2988.

[86] Lee, M., Cho, K., Park, K., Kwon, T., and Choi, Y. SCAN: scalable content routing for content-aware networking. In *Proc. IEEE Int. Conf. Commun. (ICC)* (2011), pp. 1–5.

[87] Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., Postel, J., Roberts, L. G., and Wolff, S. A brief history of the internet. *ACM SIGCOMM computer communication review 39*, 5 (2009), 22–31.

[88] Leira, L., Luís, M., and Sargento, S. Context-based caching in mobile information-centric networks. *Computer Communications 193* (2022), 214–223.

[89] Li, H., Yu, Z., Xie, K., Qiu, X., and Guo, S. Edge-feedback icn cooperative caching strategy based on relative popularity. In *Proc. 6th ICAIS'20* (2020), pp. 786–797.

[90] Li, J., Wu, H., Liu, B., Lu, J., Wang, Y., Wang, X., Zhang, Y., and Dong, L. Popularity-driven coordinated caching in named data networking. In *Proc. ACM/IEEE Symp. Architectures Netw. Commun. Syst.* (2012), pp. 15–26.

[91] Li, W., Li, Y., Wang, W., Xin, Y., and Xu, Y. A collaborative caching scheme with network clustering and hash-routing in ccn. In *Proc. IEEE PIMRC* (2016), pp. 1–7.

[92] Li, Y., Lin, T., Tang, H., and Sun, P. A chunk caching location and searching scheme in content centric networking. In *Proc. IEEE Int. Conf. Commun. (ICC)* (2012), pp. 2655–2659.

[93] Li, Y., Ouyang, S., and Lv, J. A lightweight caching decision scheme with a caching-resource-utilization-based strategy for information-centric networking. In *Proc. IEEE 49th Conference on Local Computer Networks (LCN)* (2024), pp. 1–7.

[94] Li, Y., Wang, J., and Han, R. PB-NCC: a popularity-based caching strategy with number-of-copies control in information-centric networks. *Applied Sciences 12*, 2 (2022), 653.

[95] Li, Z., and Simon, G. Time-shifted tv in content centric networks: The case for cooperative in-network caching. In *Proc. IEEE Int. Conf. Commun. (ICC)* (2011), pp. 1–6.

[96] Liao, C., Liu, X., and Zhou, H. Environment-adaptive dynamic caching for vehicular named data networks in dynamic network environments. *IEEE Transactions on Vehicular Technology 73*, 4 (2024), 5861–5871.

[97] Liu, W.-X., Zhang, J., Liang, Z.-W., Peng, L.-X., and Cai, J. Content popularity prediction and caching for icn: A deep learning approach with sdn. *IEEE access 6* (2017), 5075–5089.

[98] Lucas, J. M., and Saccucci, M. S. Exponentially weighted moving average control schemes: properties and enhancements. *Technometrics 32*, 1 (1990), 1–12.

[99] Mahmood, A., Casetti, C., Chiasserini, C. F., Giaccone, P., and Härri, J. Efficient caching through stateful sdn in named data networking. *Trans. Emerg. Telecommun. Technol. 29*, 1 (2018), e3271.

[100] Man, D., Lu, Q., Wang, H., Guo, J., Yang, W., and Lv, J. On-path caching based on content relevance in information-centric networking. *Computer Communications 176* (2021), 272–281.

[101] Meng, Y., and Ahmad, A. B. Performance measurement through caching in named data networking based internet of things. *IEEE Access 11* (2023), 120569–120584.

[102] Meng, Y., Naeem, M. A., Ali, R., and Kim, B.-S. EHCP: an efficient hybrid content placement strategy in named data network caching. *IEEE access 7* (2019), 155601–155611.

[103] Mick, T., Tourani, R., and Misra, S. MuNCC: Multi-hop neighborhood collaborative caching in information centric networks. In *Proc. of 3rd ACM ICN'16* (2016), pp. 93–101.

[104] Ming, Z., Xu, M., and Wang, D. Age-based cooperative caching in information-centric networking. In *Proc. 23rd International Conference on Computer Communication and Networks (ICCCN)* (2014), pp. 1–8.

[105] Montpetit, M.-J., Westphal, C., and Trossen, D. Network coding meets information-centric networking: an architectural case for information dis-

persion through native network coding. In *Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design-Architecture, Algorithms, and Applications* (2012), pp. 31–36.

[106] Mun, J. H., and Lim, H. Cache sharing using bloom filters in named data networking. *Journal of Network and Computer Applications 90* (2017), 74–82.

[107] Naeem, M. A., Bashir, A. K., and Meng, Y. Dynamic cluster-based cooperative cache management at the network edges in ndn-based internet of things. *Alexandria Engineering Journal 125* (2025), 297–310.

[108] Naeem, M. A., Din, I. U., Meng, Y., Almogren, A., and Rodrigues, J. J. Centrality-based on-path caching strategies in ndn-based internet of things: A survey. *IEEE Communications Surveys & Tutorials* (2024).

[109] Newman, M. E. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences 103*, 23 (2006), 8577–8582.

[110] Nguyen, X. N., Saucez, D., and Turletti, T. Efficient caching in content-centric networks using openflow. In *Proc. INFOCOM Workshops* (2013), pp. 67–68.

[111] Ong, M. D., Chen, M., Taleb, T., Wang, X., and Leung, V. C. FGPC: fine-grained popularity-based caching design for content centric networking. In *Proc. 17th ACM Int. Conf. Model. Anal. Simulat. Wireless Mobile Syst.* (2014), pp. 295–302.

[112] Pal, A., and Kant, K. A neighborhood aware caching and interest dissemination scheme for content centric networks. *IEEE Transactions on Network and Service Management 18*, 3 (2021), 3900–3917.

[113] Patil, V., Desai, H., and Zhang, L. Kua: a distributed object store over named data networking. In *Proc. 9th ACM Conf. Inf. Centric Netw.* (2022), pp. 56–66.

[114] Pavlou, G., Wang, N., Chai, W. K., and Psaras, I. Internet-scale content mediation in information-centric networks. *annals of telecommunications-annales des télécommunications 68* (2013), 167–177.

[115] Pfender, J., Valera, A., and Seah, W. K. Easy as ABC: a lightweight centrality-based caching strategy for information-centric iot. In *Proc. of 6th ACM ICN'19* (2019), pp. 100–111.

[116] Pires, S., Ribeiro, A., and Sampaio, L. N. On learning suitable caching policies for in-network caching. *IEEE Transactions on Machine Learning in Communications and Networking 2* (2024), 1076–1092.

[117] Podlipnig, S., and Böszörmenyi, L. A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR) 35*, 4 (2003), 374–398.

[118] Psaras, I., Chai, W. K., and Pavlou, G. Probabilistic in-network caching for information-centric networks. In *Proc. 2nd ACM SIGCOMM ICN Workshop* (2012), pp. 55–60.

[119] Psaras, I., Chai, W. K., and Pavlou, G. In-network cache management and resource allocation for information-centric networks. *IEEE Transactions on Parallel and Distributed Systems 25*, 11 (2014), 2920–2931.

[120] Quevedo, J., Corujo, D., and Aguiar, R. Consumer driven information freshness approach for content centric networking. In *Proc. INFOCOM WK-SHPS* (2014), pp. 482–487.

[121] Rafique, W., Hafid, A. S., and Cherkaoui, S. SoftCaching: a framework for caching node selection and routing in software-defined information centric internet of things. *Computer Networks 235* (2023), 109966.

[122] Reed, B., and Long, D. D. Analysis of caching algorithms for distributed file systems. *ACM SIGOPS Operating Systems Review 30*, 3 (1996), 12–21.

[123] Ren, J., Qi, W., Westphal, C., Wang, J., Lu, K., Liu, S., and Wang, S. MAGIC: a distributed max-gain in-network caching strategy in information-centric networks. In *Proc. IEEE INFOCOM WKSHPS* (2014), pp. 470–475.

[124] Reshadinezhad, A., Khayyambashi, M. R., and Movahedinia, N. An efficient adaptive cache management scheme for named data networks. *Future Generation Computer Systems 148* (2023), 79–92.

[125] Rezazad, M., and Tay, Y. A cache miss equation for partitioning an ndn content store. In *Proc. of the 9th Asian Internet Engineering Conference* (2013), pp. 1–8.

[126] Rezazad, M., and Tay, Y. CCndnS: a strategy for spreading content and decoupling ndn caches. In *Proc. IFIP Networking Conference (IFIP Networking)* (2015), pp. 1–9.

[127] Rodríguez-Pérez, M., Herrería-Alonso, S., Suárez-Gonzalez, A., López-Ardao, J. C., and Rodríguez-Rubio, R. Cache placement in an ndn-based leo satellite network constellation. *IEEE Transactions on Aerospace and Electronic Systems 59*, 4 (2023), 3579–3587.

[128] Rosensweig, E. J., and Kurose, J. Breadcrumbs: Efficient, best-effort content location in cache networks. In *Proc. IEEE INFOCOM 2009* (2009), pp. 2631–2635.

[129] Rossi, D., and Rossini, G. Caching performance of content centric networks under multi-path routing (and more). *Relatório técnico, Telecom ParisTech 2011* (2011), 1–6.

[130] Rossi, D., and Rossini, G. On sizing ccn content stores by exploiting topological information. In *2012 Proceedings IEEE INFOCOM Workshops* (2012), pp. 280–285.

[131] Saha, S., Lukyanenko, A., and Ylä-Jääski, A. Cooperative caching through routing control in information-centric networks. In *Proc. IEEE INFOCOM* (2013), pp. 100–104.

[132] Saha, S., Lukyanenko, A., and Ylä-Jääski, A. Efficient cache availability management in information-centric networks. *Computer Networks 84* (2015), 32–45.

[133] Saino, L., Cocora, C., and Pavlou, G. A toolchain for simplifying network simulation setup. *SimuTools 13* (2013), 82–91.

[134] Saino, L., Psaras, I., and Pavlou, G. Hash-routing schemes for information centric networking. In *Proc. 3rd ACM SIGCOMM Workshop Inf. Centric Netw. (ICN* (2013), pp. 27–32.

[135] Saino, L., Psaras, I., and Pavlou, G. Icarus: a caching simulator for information centric networking (icn). In *Proc. 7th Int. Conf. Simul. Tools Techn.* (2014), pp. 66–75.

[136] Saltarin, J., Braun, T., Bourtsoulatze, E., and Thomos, N. PopNetCod: a popularity-based caching policy for network coding enabled named data networking. In *Proc. of IFIP Networking Conference & Workshops* (2018), pp. 271–279.

[137] Saroiu, S., Gummadi, K. P., Dunn, R. J., Gribble, S. D., and Levy, H. M. An analysis of internet content delivery systems. *ACM SIGOPS Operating Systems Review 36*, SI (2002), 315–327.

[138] Sato, Y., Ito, Y., and Koga, H. Hash-based cache distribution and search schemes in content-centric networking. *IEICE TRANSACTIONS on Information and Systems 102*, 5 (2019), 998–1001.

[139] Serhane, O., Yahyaoui, K., Nour, B., and Moungla, H. Energy-aware cache placement scheme for iot-based icn networks. In *Proc. IEEE Int. Conf. Commun. (ICC)* (2021), pp. 1–6.

[140] Sharif, S., Moghaddam, M. H. Y., and Seno, S. A. H. Adaptive cache content placement for software-defined internet of things. *Future Generation Computer Systems 136* (2022), 34–48.

[141] Silva, A., Araujo, I., Linder, N., and Klautau, A. Name popularity algorithm: a cache replacement strategy for ndn networks. *Journal of Communication and Information Systems 34*, 1 (2019), 206–214.

[142] Siris, V. A., Vasilakos, X., and Polyzos, G. C. Efficient proactive caching for supporting seamless mobility. In *Proc. IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014* (2014), pp. 1–6.

[143] Song, J., Yang, Y., Jin, W., and Song, T. Adaptive segmented subscription for efficient data dissemination in vehicular named data networks. *IEEE Transactions on Network and Service Management 21*, 4 (2024), 4466–4479.

[144] Sourlas, V., Flegkas, P., and Tassiulas, L. Cache-aware routing in information-centric networks. In *Proc. IFIP/IEEE International Symposium on Integrated Network Management* (2013), pp. 582–588.

[145] Sourlas, V., Psaras, I., Saino, L., and Pavlou, G. Efficient hash-routing and domain clustering techniques for information-centric networks. *Computer Networks 103* (2016), 67–83.

[146] Spring, N., Mahajan, R., and Wetherall, D. Measuring isp topologies with rocketfuel. In *Proc. ACM SIGCOMM* (2002), pp. 133–145.

[147] Spring, N., Mahajan, R., Wetherall, D., and Anderson, T. Measuring isp topologies with rocketfuel. *IEEE/ACM Transactions on networking 12*, 1 (2004), 2–16.

[148] Tan, B., and Massoulié, L. Optimal content placement for peer-to-peer video-on-demand systems. *IEEE/ACM transactions on networking 21*, 2 (2012), 566–579.

[149] Tarnoi, S., Kumwilaisak, W., Suppakitpaisarn, V., Fukuda, K., and Ji, Y. Adaptive probabilistic caching technique for caching networks with dynamic content popularity. *Computer Communications 139* (2019), 1–15.

[150] Tarnoi, S., Suksomboon, K., Kumwilaisak, W., and Ji, Y. Performance of probabilistic caching and cache replacement policies for content-centric networks. In *Proc. 39th IEEE Conf. Local Comput. Netw. (LCN)* (2014), pp. 99–106.

[151] Thomas, Y., Xylomenos, G., Tsilopoulos, C., and Polyzos, G. C. Object-oriented packet caching for icn. In *Proc. of 2nd ACM ICN'15* (2015), pp. 89–98.

[152] Tsai, P.-H., Zhang, J.-B., and Tsai, M.-H. An efficient probe-based routing for content-centric networking. *Sensors 22*, 1 (2022), 341.

[153] Wang, C., Chen, C., Pei, Q., Lv, N., and Song, H. Popularity incentive caching for vehicular named data networking. *IEEE Transactions on Intelligent Transportation Systems 23*, 4 (2022), 3640–3653.

[154] Wang, J. A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communication Review 29*, 5 (1999), 36–46.

[155] Wang, J. M., and Bensaou, B. Progressive caching in ccn. In *Proc. IEEE GLOBECOM* (2012), IEEE, pp. 2727–2732.

[156] Wang, J. M., Zhang, J., and Bensaou, B. Intra-AS cooperative caching for content-centric networks. In *Proc. 3rd ACM SIGCOMM Workshop Inf. Centric Netw. (ICN)* (2013), pp. 61–66.

[157] Wang, L., Bayhan, S., Ott, J., Kangasharju, J., Sathiaseelan, A., and Crowcroft, J. Pro-Diluvian: understanding scoped-flooding for content discovery in information-centric networking. In *Proc. of the 2nd ACM ICN* (2015), pp. 9–18.

[158] Wang, S., Bi, J., Wu, J., and Vasilakos, A. V. CPHR: in-network caching for information-centric networking with partitioning and hash-routing. *IEEE/ACM Transactions on Networking 24*, 5 (2015), 2742–2755.

[159] Wang, W., Sun, Y., Guo, Y., Kaafar, D., Jin, J., Li, J., and Li, Z. CRCache: exploiting the correlation between content popularity and network topology information for icn caching. In *Proc. IEEE Int. Conf. Commun. (ICC)* (2014), pp. 3191–3196.

[160] Wang, X., and Wu, G. Information-centric wireless sensor networks based multimedia content caching and dissemination. *IEEE Sensors Journal 24*, 13 (2024), 21549–21557.

[161] Wang, Y., Xu, M., and Feng, Z. Hop-based probabilistic caching for information-centric networks. In *Proc. IEEE GLOBECOM* (2013), pp. 2102–2107.

[162] Wessels, D. *Web caching.* " O'Reilly Media, Inc.", 2001.

[163] Wissingh, B., Wood, C. A., Afanasyev, A., Zhang, L., Oran, D. R., and Tschudin, C. RFC 8793: Information-Centric Networking (ICN): Content-Centric Networking (CCNx) and Named Data Networking (NDN) terminology, 2020.

[164] Wolman, A., Voelker, M., Sharma, N., Cardwell, N., Karlin, A., and Levy, H. M. On the scale and performance of cooperative web proxy caching. In *Proc. of 17th ACM SOSP* (1999), pp. 16–31.

[165] Wu, H., Li, J., Pan, T., and Liu, B. A novel caching scheme for the backbone of named data networking. In *Proc. IEEE Int. Conf. Commun. (ICC)* (2013), pp. 3634–3638.

[166] Wu, H., Xu, Y., and Li, J. PTF: popularity-topology-freshness-based caching strategy for icn-iot networks. *Computer Communications 204* (2023), 147–157.

[167] Wu, Q., Li, Z., and Xie, G. Codingcache: multipath-aware ccn cache with network coding. In *Proc. ACM SIGCOMM Workshop Inf. Centric Netw.* (2013), pp. 41–42.

[168] Xu, R., Di, X., Chen, J., Qi, H., and Zhao, L. Community division-based content distribution in information-centric satellite networks: An efficient approach for remote sensing. *IEEE Internet of Things Journal* (2024).

[169] Xylomenos, G., Ververidis, C. N., Siris, V. A., Fotiou, N., Tsilopoulos, C., Vasilakos, X., Katsaros, K. V., and Polyzos, G. C. A survey of information-centric networking research. *IEEE communications surveys & tutorials 16*, 2 (2013), 1024–1049.

[170] Yan, H., Gao, D., Su, W., Foh, C. H., Zhang, H., and Vasilakos, A. V. Caching strategy based on hierarchical cluster for named data networking. *IEEE Access 5* (2017), 8433–8443.

[171] Yang, Y., Song, T., and Zhang, B. OpenCache: a lightweight regional cache collaboration approach in hierarchical-named icn. *Computer Communications 144* (2019), 89–99.

[172] Yao, L., Xu, X., Deng, J., Wu, G., and Li, Z. A cooperative caching scheme for vccn with mobility prediction and consistent hashing. *IEEE Transactions on Intelligent Transportation Systems 23*, 11 (2022), 20230–20242.

[173] Yoshida, M., Ito, Y., Sato, Y., and Koga, H. A cluster-based cache distribution scheme in content-centric networking. In *Proc. 5th ACM Conf. Inf. Centric Netw.* (2018), pp. 196–197.

[174] Yoshida, M., Ito, Y., Sato, Y., and Koga, H. Popularity-aware dynamic clustering scheme for distributed caching in icn. In *Proc. 9th ACM Conf. Inf. Centric Netw.* (2022), pp. 192–193.

[175] Yoshida, M., Ito, Y., Sato, Y., and Koga, H. PopDCN: popularity-aware dynamic clustering scheme for distributed caching in icn. *IEICE Transactions on Communications 107*, 5 (2024), 398–407.

[176] Yu, M., and Li, R. Dynamic popularity-based caching permission strategy for named data networking. In *Proc. 22nd CSCWD'18* (2018), pp. 576–581.

[177] Yu, Y., Gan, Y., Sarda, N., Tsai, L., Shen, J., Zhou, Y., Krishnamurthy, A., Lai, F., Levy, H., and Culler, D. IC-Cache: efficient large language model serving via in-context caching. In *Proc. of the ACM SIGOPS 31st Symposium on Operating Systems Principles* (2025), pp. 375–398.

[178] Zhang, B. Fault Management in NDN. `https://named-data.net/wp-content/uploads/2014/05/FIA-2013-NDN-Fault-Mgmt-11-14-2013.pdf`, 2013. Last accessed on May 26, 2026.

[179] Zhang, G., Li, Y., and Lin, T. Caching in information centric networking: A survey. *Computer networks 57*, 16 (2013), 3128–3141.

[180] Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., claffy, k., Crowley, P., Papadopoulos, C., Wang, L., and Zhang, B. Named data networking. *ACM SIGCOMM Computer Communication Review 44*, 3 (2014), 66–73.

[181] Zhang, M., Luo, H., and Zhang, H. A survey of caching mechanisms in information-centric networking. *IEEE Communications Surveys & Tutorials 17*, 3 (2015), 1473–1499.

[182] Zhang, R., Liu, J., Huang, T., and Xie, R. Popularity based probabilistic caching strategy design for named data networking. In *Proc. IEEE INFOCOM WKSHPS* (2017), pp. 476–481.

[183] Zhang, T., Xu, X., Zhou, L., Jiang, X., and Loo, J. Cache space efficient caching scheme for content-centric mobile ad hoc networks. *IEEE Systems Journal 13*, 1 (2019), 530–541.

[184] Zhang, Y., Cui, L., Wang, W., and Zhang, Y. A survey on software defined networking with multiple controllers. *Journal of Network and Computer Applications 103* (2018), 101–118.

[185] Zhang, Z., Lung, C.-H., Lambadaris, I., and St-Hilaire, M. Iot data lifetime-based cooperative caching scheme for icn-iot networks. In *Proc. IEEE Int. Conf. Commun. (ICC)* (2018), pp. 1–7.

[186] Zhang, Z., Lung, C.-H., St-Hilaire, M., and Lambadaris, I. An SDN-based caching decision policy for video caching in information-centric networking. *IEEE Transactions on Multimedia 22*, 4 (2020), 1069–1083.

[187] Zhang, Z., Lung, C.-H., St-Hilaire, M., and Lambadaris, I. Smart proactive caching: Empower the video delivery for autonomous vehicles in icn-based networks. *IEEE Transactions on Vehicular Technology 69*, 7 (2020), 7955–7965.

[188] Zhang, Z., Lung, C.-H., Wei, X., Chen, M., Chatterjee, S., and Zhang, Z. In-network caching for icn-based iot (icn-iot): A comprehensive survey. *IEEE Internet of Things Journal 10*, 16 (2023), 14595–14620.

[189] Zhang, Z., Wei, X., Lung, C.-H., and Zhao, Y. iCache: an intelligent caching scheme for dynamic network environments in icn-based iot networks. *IEEE Internet of Things Journal 10*, 2 (2022), 1787–1799.

[190] Zheng, Q., Kan, Y., Chen, J., Wang, S., and Tian, H. A cache replication strategy based on betweenness and edge popularity in named data networking. In *Proc. IEEE Int. Conf. Commun. (ICC)* (2019), pp. 1–7.

[191] Zhu, X., Wang, J., Wang, L., and Qi, W. Popularity-based neighborhood collaborative caching for information-centric networks. In *Proc. IEEE 36th IPCCC* (2017), pp. 1–8.