EeNCache : Neighborhood Cooperative Content Caching and Delivery in Named Data Networks

Pankaj Chaudhary and Neminath Hubballi
Indian Institute of Technology Indore, India
{phd2001201004, neminath}@iiti.ac.in

***Abstract*—Named Data Networking (NDN) offers in-network caching which helps to improve content delivery efficiency and minimizes bandwidth usage. However, cache being a prized commodity needs to be used judiciously. Hence, many studies propose different caching techniques aimed at improving content delivery. However, achieving a balance between good performance and overhead in real-world scenarios remains a challenge. In this paper, we outline a lightweight neighborhood cooperative searching and caching technique for NDN, named `EeNCache`. It cooperates with routers in the neighborhood to base its caching and searching decisions. This cooperation is achieved through routers periodically sharing their cache catalogue with their neighbors. This catalogue is maintained in the form of a Bloom filter data structure. In addition, `EeNCache` also uses a simple mechanism to cache popular content. The performance of `EeNCache` is evaluated on a large-scale network topology, comparing it with traditional and recent caching techniques using the Python-based Icarus simulator. Our findings demonstrate that cooperative caching yields better results in terms of cache hit ratio and content access time.**

***Keywords*—Next-generation Networks, Named Data Networks, Content Delivery, Caching, Bloom Filter**

I. Introduction

Designing efficient caching techniques has gained attention in recent years, particularly in the context of next-generation networks. It involves storing content closer to consumers for faster retrieval. Information-Centric Networking (ICN) is a new network architecture which inherently supports in-network caching. Many ICN projects [7] have been initiated globally to address the challenges faced by the current Internet architecture. TCP/IP-based networking architecture is host-centric and very inefficient in content delivery, warranting a transition from the host-centric model to a content-centric one offered by ICN. Named Data Networking (NDN) [20] is one of the most popular and currently active ICN projects which allows multiple users to reuse the cached content. This is achieved by maintaining data structures such as the Content Store (CS) and the Pending Interest Table (PIT) at each router. The CS is used for storing content in the network, while the PIT helps track incoming and outgoing interfaces, enabling the same content to be utilized by multiple users. NDN further reduces the burden on the control plane, as its data plane is intelligent enough to handle faults in the network. Information exchange in NDN takes place using two types of packets namely Interest and Data packets. An Interest packet is generated by the consumer when requesting content such as video, music, or images. In response, the content provider prepares a Data packet that carries the requested content. These packets travel across multiple hops between the content consumer and the provider. During the transmission of these content packets, caching decisions are done by the intermediate routers which helps in reducing the number of hops for subsequent requests and also in alleviating the load on the source.

Recent studies [3], [6], [8], [13], [15] have explored various caching strategies in NDN to enhance content delivery by storing popular content closer to consumers. These methods generally fall into either autonomous decision-making by individual routers or approaches where routers collaborate to share cache information. Cooperative caching allows routers to collaborate and use content outside the transmission path, allowing better storage utilization. However, the cooperative approach adds a communication overhead on the routers due to the need for additional coordination and exchange of cache information with neighboring routers. We aim to make use of the content available in the neighboring routers to achieve a high cache hit ratio and minimize content delivery time while maintaining low communication overhead. To this end, we propose `EeNCache`, a cooperative content searching and caching mechanism using the memory-efficient Bloom filter data structure. Bloom filters are used to periodically share cache information with neighboring routers. Since the Bloom filter is a probabilistic model, it may occasionally result in false positives, indicating content availability where it does not exist. However, `EeNCache` simultaneously explores both on-path and off-path routers, minimizing the impact of false positives. Further, `EeNCache` considers the frequency of requests for content before caching it for efficient storage utilization. In summary, we make the following contributions in this paper.
(i) A cooperative searching and caching method is designed using Bloom filters to share cache catalogue information with neighbors, reducing communication overhead and redundancy.
(ii) A lightweight popularity estimation technique is presented to optimize the limited caching capacity of routers.
(iii) The performance of `EeNCache` is evaluated using an open-source simulator, focusing on cache hit ratio and content access time in a large-scale topology.

II. Literature Review

Traditional caching approaches in NDN, such as Cache Everything Everywhere (CEE) [10] and Leave Copy Down (LCD) [12], use predetermined policies to decide where to cache content. In CEE, routers cache all content that passes through them, while in LCD, only the router located one hop down from the serving node caches the content. With

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

each cache hit, the content moves closer to the end-users. However, these strategies do not optimize router cache storage and neglect user demand, resulting in lower cache hit ratios and diversity. Recent caching techniques prioritize content placement by considering dynamic factors such as user demands [15], content freshness [3], router connectivity [11], distance from the user [13], cache occupancy [9], etc.

Different methods use different techniques to estimate the popularity of content. For example, Yu *et al.* [19] proposed DPCP, a caching method that dynamically estimates the popularity threshold based on user demands to determine whether a particular chunk is beneficial for caching. In [2], the authors designed a caching technique in which content is eligible for caching based on a small predefined threshold that helps quickly bring popular content closer to the consumer. Man *et al.* [14] designed a caching technique based on the correlation between the content that needs to be cached and the content already available in the router's cache, as well as the position of the router in the content delivery path. In [3], [15], caching techniques consider both the freshness of the content and its popularity when making caching decisions. The content freshness parameter is beneficial for applications like the Internet of Things (IoT), where the lifecycle of content is crucial for maximizing the advantages of caching. In [11], the authors proposed a caching technique called PT-Cache, which argues that caching content based on the closeness centrality of routers, along with content popularity, helps improve the cache hit ratio and content access time. Similarly, works [9], [13] propose a caching technique where caching decisions are made by the router based on the available space and popularity.

Another set of works tries to optimize cache utilization with cooperation. For example, in our previous work [6], we proposed a cooperative technique called eNCache, where each router situated between the consumer and provider simultaneously queries packets to its N-hop neighborhood until the content is found. In the content delivery direction, routers also collaborate to make caching decisions. The reactive querying of this method causes a huge number of message exchanges in the network. To establish cooperation, [8] uses network diameter to designate a few nodes as leaders, which are responsible for collecting the global popularity of content for caching. In [16], the authors propose a caching method that uses the Bloom filter to share cache information among neighbors, thereby reducing content redundancy in the neighborhood. This strategy considers the case of false positives when the face for incoming and outgoing Interests is the same. If a false positive is detected, the request is forwarded to the next hop by consulting the routing table. However, in `EeNCache`, we combine cooperation and popularity with content availability in the neighborhood for caching and searching decisions, significantly reducing the overhead.

III. Proposed Caching Technique

In this section, we present the working of `EeNCache`.

Motivation: In our previous work, we designed eNCache [6], a cooperative searching and caching mechanism for NDN. The performance of eNCache is notable as it uses an N-hop neighborhood searching mechanism without introducing off-path delays. eNCache forwards packets to all N-hop neighbors simultaneously without waiting to check whether the content is available in off-path neighbors, since the request is always forwarded towards the original source. However, we observed that eNCache resulted in unnecessary flooding of upstream traffic, leading to high signaling overhead in large-scale topologies. To overcome the limitations of eNCache, we design `EeNCache`, which also transmits packets simultaneously to off-path and on-path neighbors. `EeNCache` uses a Bloom filter data structure which keeps track of which neighbor has the content. This helps to reduce the unnecessary flooding of packets to off-path neighbors. Routers periodically exchange their Bloom filters with neighboring routers to obtain cache information about neighbors. If the Bloom filter suggests that the content is present in any neighboring router, the request is sent to that neighbor to retrieve the content.

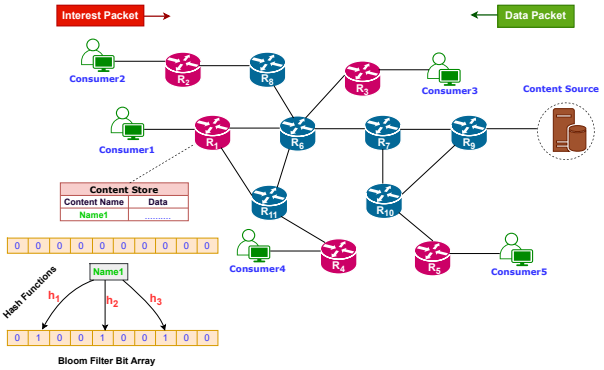


Fig. 1: `EeNCache` Reference Architecture

A. Neighborhood Collaboration

The capacity of a router is limited, so sharing information among routers helps better utilize this limited resource by reducing the caching of the same content across multiple routers in a network and also improves content delivery efficiency. On-path caching does not take advantage of content available outside the transmission path, even if that router may be closer to the consumer. Searching for content off the main path without prior information about off-path routers can result in increased delays and overhead. We use a Bloom filter [4] data structure to check for content availability in off-path routers. A Bloom filter is a fast and memory-efficient data structure used for membership testing to determine whether content is available in a neighbor. It uses a bit array of size `m` and applies `k` different hash functions to map content to specific positions in the bit array as shown in Fig. 1. Due to its compact size, it is efficient to share with neighbors instead of sending the entire CS table. In Fig. 1, a Bloom filter with `m = 10` bits and `k = 3` initially has all bits set to 0. After applying the three hash functions to `Name1`, the resulting indices are 1, 4, and 7, which set the bits at these positions to 1. If any of the bits are 0, it indicates that the content is not available in the neighborhood, as a Bloom filter provides no false negatives, whereas if the bits are all 1, the content might be available but not guaranteed.

`EeNCache` uses this Bloom filter for both routing requests

towards the potential content provider and caching content in the downstream direction. Each router maintains a Bloom filter that represents the contents stored in its cache as well as the caches of its neighboring routers. Routers exchange cache information with N-hop neighbors by periodically sharing their Bloom filters at time `T` intervals. After receiving Bloom filters from neighboring routers, each router makes an informed decision on request forwarding and caching. In the next subsections, we discuss how routers use the Bloom filter for Interest packet forwarding and caching content.

B. Interest Forwarding

When a router receives an Interest packet from a downstream neighbor, it first checks its CS to see if the content is available. If the content is found, it is returned; otherwise, the router decides how to forward the packet towards the potential provider. By default, the packet is forwarded towards the source using the best route strategy. Each router consults its local routing table to send the packet to the next upstream router. Along with the next on-path router, `EeNCache` also explores off-path routers while forwarding the packet to the original source. For example, in Fig. 1, when Consumer1 requests `Name1`, and it is available at R_{11} and R_9, the standard forwarding approach retrieves the content from R_9, which takes four hops, whereas `EeNCache` retrieves the content from R11, located outside the shortest path, taking only two hops.

When the CS lookup fails at an on-path router, it must forward the packet to the next hop. Before doing so, the router checks the Bloom filter; if a match is found, the request is forwarded not only to the next hop but also to all matching off-path interfaces. If the Bloom filter lookup returns false, the request is forwarded only to the next hop router based on the routing table. This process is repeated at each on-path router located between the consumer and provider until the content is found. `EeNCache` does not wait for content retrieval from off-path routers, as it simultaneously forwards requests toward the source using the best route strategy. This approach ensures that no additional delay is introduced while searching for content in off-path routers. For example, in Fig. 1, if the Bloom filter of R_1 indicates that content is available in the neighbor R_{11}, R_1 forwards the request to both R_6 and R_{11} simultaneously. If the content is not found at R_{11}, this means a false positive case of the Bloom filter. However, this does not impact `EeNCache`, as R_1 does not wait for the response from R_{11}; the request has already been forwarded to R_6, the next router along the path to the original source.

Furthermore, each router locally estimates the frequency of requests by maintaining a data structure to determine the popularity of content. When it receives a request, it increments the `InterestCount`. This local table at each router helps differentiate between popular and unpopular content while making caching decisions. Algorithm 1 outlines the details of how requests are forwarded and how the local frequency of requests is measured at each router R_i.

C. Content Caching

When an Interest packet successfully reaches the provider, the provider responds with the Data packet. The routers responsible for forwarding the content towards the consumer then decide whether to cache it or forward it to the next hop based on the PIT entry. Caching all content in every router quickly fills up storage, as some content may only be requested once or twice and then not requested again for a long time. To address this, `EeNCache` uses a predetermined threshold (P_{th}) value to make caching decisions at non-edge routers, as these routers share paths with multiple consumers. We chose a fixed threshold value over a dynamic one because it does not require any bookkeeping, allowing for faster decisions while also helping to prevent storage saturation. Algorithm 2 outlines the details of Data packet processing at each router R_i.

Algorithm 1: Interest Packet Processing at Router R_i

```
Name ← Content Request at R_i
R_j ← NextHopOf(R_i)
R_n ← {r ∈ N-Hop Neighborhood of R_i | r ≠ R_j}
R_f ← MatchingInterfaceOf(Name) ∈ R_n
if Name ∈ CacheOf(R_i) then
    Return Data to the requester
    InterestCount++
else if BloomFilter.Lookup(Name) == True then
    Send Interest to R_j and R_f
    if Name ∈ CacheOf(R_j or R_f) then
        Return Data to the requester
        InterestCount++
    end if
else
    Forward Interest to R_j // Bloom filter lookup is false
    InterestCount++
end if
```

Algorithm 2: Data Packet Processing at Router R_i

```
Name ← Data Packet Arrives at R_i
if R_i is an Edge Router then
    Cache Name at R_i // If Cache is Full, Use Cache Replacement Policy
else
    if FreeSpaceOf(R_i) ≥ SizeOf(Name) then
        if InterestCountOf(Name) ≥ P_th then
            Cache Name in R_i
        end if
    else
        if MatchInterfaceOf(Name) ∉
        BloomFilter.Lookup(Name) then
            if InterestCountOf(Name) ≥ P_th then
                Cache Name in R_i // Cache is Full, Use Cache Replacement Policy
            else
                Forward Data to the next hop without caching
            end if
        end if
    end if
end if
```

`EeNCache` follows these steps for making caching decisions:
(i) If the router is an edge router with respect to the consumer that generated the Interest, the router will always cache the content. If the cache is full, it will use a replacement policy to store the new content. Caching at the edge helps minimize content retrieval time.

(ii) If the router is not an edge router and has sufficient available space, it will cache the content if the request frequency is higher than P_{th}. This helps prevent quick saturation of the router's storage.
(iii) If the router is a non-edge router but does not have sufficient space, it will use the replacement policy to cache the new content if the caching criteria are met; otherwise, it will forward the content to the next hop without caching. The new content should not be available in the CS of neighboring routers. After performing a Bloom filter lookup, if the Bloom filter indicates the content is unavailable, the router will check if the request frequency for the new content exceeds P_{th}. If so, it will cache the content; otherwise, it will ignore it.

It is worth noting that by consulting the Bloom filter for both searching and caching, `EeNCache` reduces the overhead of communication significantly.

IV. Experiments and Evaluation

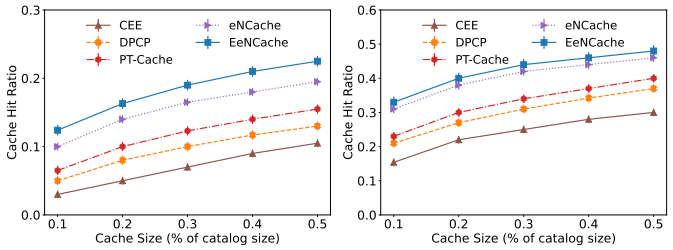
In the following subsections, we compare `EeNCache`'s performance with four other ICN/NDN caching techniques: CEE [10], DPCP [19], PT-Cache [11], and eNCache [6].

A. Simulation Setup

We use the Python-based open-source simulator Icarus [17], which is specifically designed for the performance evaluation of caching techniques. To evaluate `EeNCache`, we used a large-scale topology, Exodus (US), from the RocketFuel [18] ISP map in Icarus. This topology consists of 201 nodes (routers) and 434 edges, out of which 32 nodes are degree-1 nodes. We installed a consumer stack on 30 nodes and a source stack on two nodes. The source and consumer nodes are connected to content routers with a degree greater than one. The link delay between each pair of nodes is set to 5 milliseconds. In the simulation, the content catalog consists of 5×10^4 items, and the cache size of each router varies between 0.1% to 0.5% of the catalog size, assuming all content items are of equal size. Least Recently Used (LRU) is used as the replacement policy, as it is faster for performing insertions and deletions and is claimed in many NDN caching studies [3], [13] to provide reasonable performance. The popularity of content is modeled using a Zipf [5] distribution with a skewness parameter α set to 0.8 and 1.0. Consumer request arrivals follow a Poisson distribution with an average rate of 10 requests per second. The first 5×10^4 requests are generated for warm-up to stabilize the network, followed by 2×10^5 requests for performance measurements. For `EeNCache`, we used the PyBloom [1] module to implement a Bloom filter for sharing cache information with neighbors. The parameters for the Bloom filter are configured to approximately 90 bytes for `m`, 9 for `k`, and 0.001 for false positive probability. These values are adjustable based on the size of the cache and the number of items that need to be stored in the Bloom filter. The time interval for exchanging cache summaries within the neighborhood is set to one minute. The threshold parameter (P_{th}) is set to 2 for caching decisions. For performance comparison, the off-path neighborhood exploration for both eNCache and `EeNCache` is limited to 1-hop. Each experiment was run five times, and the average with a 95% confidence interval was used for performance analysis.

B. Evaluation Results

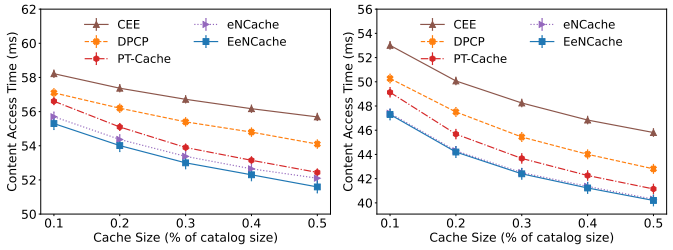
We assess the performance of `EeNCache` with two parameters namely cache hit ratio and latency of content access.
1. Cache Hit Ratio: It represents the proportion of total requests served by the routers to the total number of requests made by the consumers. Fig. 2 shows the cache hit ratio of five caching techniques as cache size varies, with α values of 0.8 and 1.0, respectively.



(a) $\alpha = 0.8$ (b) $\alpha = 1.0$

Fig. 2: Comparison of Cache Hit Ratios for Different Caching Techniques.

As cache size and α increase, the cache hit ratio improves for all strategies. Cooperative off-path searching strategies eNCache and `EeNCache` perform better than non-cooperative schemes CEE, DPCP, and PT-Cache. However, due to the consideration of user demands along with neighbor cooperation, the cache hit ratio of `EeNCache` is up to 23.5% higher than eNCache when $\alpha = 0.8$ and up to 6.5% higher when $\alpha = 1.0$. As α increases, requests become more concentrated on fewer content items, leading to a higher hit ratio across all strategies. It is worth noting that, regardless of cache size and the popularity parameter (α), `EeNCache` outperforms the other four caching techniques.
2. Content Access Time: This metric measures the overall time required for Interest packets to travel to the providers and for Data packets to return to the consumers. Figs. 3a and 3b demonstrate the content access time of five caching techniques with α values of 0.8 and 1.0, respectively.



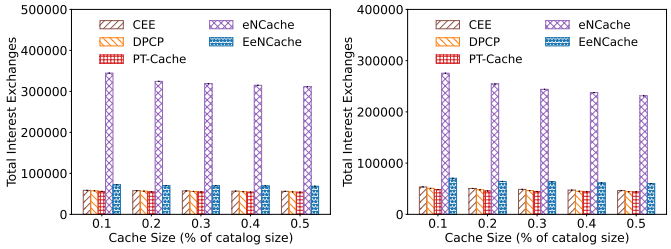
(a) $\alpha = 0.8$ (b) $\alpha = 1.0$

Fig. 3: Content Access Time Comparison for Different Caching Techniques.

CEE experiences the highest retrieval time compared to other methods because it caches content in all routers along the path, resulting in poor utilization of router storage. DPCP performs better than CEE by considering content popularity and avoiding redundant caching in downstream neighbors during content forwarding. PT-Cache achieves better performance

than both CEE and DPCP by caching based on closeness centrality and request frequency. The cooperative off-path content searching in eNCache and EeNCache lead to improved content access time, as content can often be retrieved from off-path neighbors without reaching the farther on-path provider or source. EeNCache further reduces content access time slightly compared to eNCache by making caching decisions based on request frequency.

3. Overhead: Overhead represents the total number of Interest messages transmitted in the network to retrieve Data packets. In non-cooperative forwarding strategies, Interest packets are only transmitted between the consumer and provider. However, in cooperative forwarding strategies, additional Interest messages may be transmitted to involve neighboring nodes for cooperation. For the overhead analysis, a total of 10^4 requests



(a) $\alpha = 0.8$ (b) $\alpha = 1.0$

Fig. 4: Overhead Analysis of Various Caching Techniques.

are generated in the simulation. Figs. 4a and 4b illustrate the overhead of five caching techniques with popularity parameter α set at 0.8 and 1.0, respectively. CEE, DPCP, and PT-Cache do not use cooperative searching for content, resulting in lower message transmission. On the other hand, eNCache and EeNCache use cooperative searching within a 1-hop neighborhood, generating additional Interest packets for off-path queries. eNCache results in high message transmission in complex network topologies because each on-path router reactively sends Interest packets to all 1-hop neighbors. The proposed EeNCache strategy achieves cooperation with the help of a Bloom filter, which reduces the number of messages transmitted to off-path neighbors. Despite the potential for false positives generated by the Bloom filter, EeNCache still achieves better performance compared to eNCache. It is worth noting that EeNCache adds slightly more overhead than non-cooperative approaches to achieve better performance in terms of cache hit ratio and content access time.

V. Conclusion

This paper proposes a caching technique to solve the overhead problem in cooperative caching for NDN. The proposed EeNCache method involves cooperative searching and caching with the help of Bloom filters. Routers periodically share their Bloom filters with directly connected neighboring routers. Additionally, EeNCache considers the frequency of requests while making caching decisions. The performance of EeNCache is evaluated on a large-scale, real-world topology and compared with classical caching algorithm CEE, popularity-based techniques DPCP and PT-Cache, and cooperative technique eNCache. Simulation results confirm that EeNCache achieves a better cache hit ratio and content delivery with minimal overhead. EeNCache achieves significantly lower overhead than the cooperative technique eNCache. Future work will focus on further improving caching performance and overhead by exploring alternative Bloom filter mechanisms to reduce false positives and considering the dynamic characteristics of the network.

References

[1] PyBloom: A Probabilistic data structure. Online. Available: https://github.com/jaybaird/python-bloomfilter. Last accessed on November 30, 2024.

[2] S. Alduayji, A. Belghith, A. Gazdar, and S. Al-Ahmadi. Pf-edgecache: Popularity and freshness aware edge caching scheme for ndn/iot networks. *Pervasive and Mobile Computing*, 91:101782, 2023.

[3] M. Amadeo, C. Campolo, G. Ruggeri, and A. Molinaro. Beyond edge caching: Freshness and popularity aware IoT data caching via NDN at internet-scale. *IEEE Transactions on Green Communications and Networking*, 6(1):352–364, 2022.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *IEEE INFOCOM'99*, volume 1, pages 126–134, 1999.

[6] P. Chaudhary, N. Hubballi, and S. G. Kulkarni. eNCache: improving content delivery with cooperative caching in named data networking. *Computer Networks*, page 110104, 2023.

[7] M. Conti, A. Gangwal, M. Hassan, C. Lal, and E. Losiouk. The road ahead for networking: A survey on icn-ip coexistence solutions. *IEEE Communications Surveys & Tutorials*, 22(3):2104–2129, 2020.

[8] N. Hubballi and P. Chaudhary. CPCache: cooperative popularity based caching for named data networks. In *Proc. 38th Int. Conf. Inf. Netw. (ICOIN)*, pages 379–384, 2024.

[9] N. Hubballi, P. Chaudhary, and S. G. Kulkarni. PePC: popularity based early predictive caching in named data networks. In *Proc. IEEE 21st Annu. Consum. Commun. Netw. Conf. (CCNC)*, pages 478–483, 2024.

[10] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proc. ACM CoNEXT*, pages 1–12, 2009.

[11] M. Koide, N. Matsumoto, and T. Matsuzawa. Caching method for information-centric ad hoc networks based on content popularity and node centrality. *Electronics*, 13(12):2416, 2024.

[12] N. Laoutaris, H. Che, and I. Stavrakakis. The LCD interconnection of LRU caches and its analysis. *Performance Evaluation*, 63(7):609–634, 2006.

[13] Y. Li, S. Ouyang, and J. Lv. A lightweight caching decision scheme with a caching-resource-utilization-based strategy for information-centric networking. In *Proc. IEEE 46th Conf. Local Comput. Netw. (LCN)*, pages 1–7, 2024.

[14] D. Man, Q. Lu, H. Wang, J. Guo, W. Yang, and J. Lv. On-path caching based on content relevance in information-centric networking. *Computer Communications*, 176:272–281, 2021.

[15] Y. Meng and A. B. Ahmad. Performance measurement through caching in named data networking based internet of things. *IEEE Access*, 11:120569–120584, 2023.

[16] J. H. Mun and H. Lim. Cache sharing using a bloom filter in named data networking. In *Proc. of the 2016 Symposium on Architectures for Networking and Communications Systems*, pages 127–128, 2016.

[17] L. Saino, I. Psaras, and G. Pavlou. Icarus: a caching simulator for information centric networking (ICN). In *Proc. 7th Int. Conf. Simul. Tools Techn.*, pages 66–75, 2014.

[18] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002.

[19] M. Yu and R. Li. Dynamic popularity-based caching permission strategy for named data networking. In *Proc. IEEE 22nd CSCWD'18*, pages 576–581, 2018.

[20] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.