Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 3791859, 13 pages
https://doi.org/10.1155/2021/3791859

Research Article

Popularity-Aware In-Network Caching for Edge Named Data Network

Jiliang Yin,[1] **Congfeng Jiang**,[1] **Hidetoshi Mino**,[2] **and Christophe Cérin**[3]

[1]*School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China*
[2]*Department of Computer Science and Engineering, University of Yamanashi, Yamanashi 400-0013, Japan*
[3]*LIPN UMR CNRS 7030, Université Sorbonne Paris Nord, Villetaneuse F-93430, France*

Correspondence should be addressed to Congfeng Jiang; cjiang@hdu.edu.cn

Received 1 May 2021; Revised 20 July 2021; Accepted 16 August 2021; Published 31 August 2021

Academic Editor: Xiaoxian Yang

The traditional centralized network architecture can lead to a bandwidth bottleneck in the core network. In contrast, in the information-centric network, decentralized in-network caching can alleviate the traffic flow pressure from the network center to the edge. In this paper, a popularity-aware in-network caching policy, namely, *Pop*, is proposed to achieve an optimal caching of network contents in the resource-constrained edge networks. Specifically, *Pop* senses content popularity and distributes content caching without adding additional hardware and traffic overhead. We conduct extensive performance evaluation experiments by using *ndnSIM*. The experiments showed that the *Pop* policy achieves 54.39% cloud service hit reduction ratio and 22.76% user request average hop reduction ratio and outperforms other policies including Leave Copy Everywhere, Leave Copy Down, Probabilistic Caching, and Random choice caching. In addition, we proposed an ideal caching policy (*Ideal*) as a baseline whose popularity is known in advance; the gap of *Pop* and *Ideal* in cloud service hit reduction ratio is 4.36%, and the gap in user request average hop reduction ratio is only 1.47%. More simulation results further show the accuracy of *Pop* in perceiving popularity of contents, and *Pop* has good robustness in different request scenarios.

1. Introduction

With the rapid expansion of the Internet, diversified Internet content services are deployed based on the *publish-subscribe* [1] service model. This model enables the service provider to publish or share content in a central server, while all its subscribers can access the content independently through the distributed networking connections. However, the *publish-subscribe* model can lead to a bandwidth bottleneck in the backbone network in the traditional Internet protocol- (IP-) based network architecture. For a large-scale content service, the subscribers' considerable data transmission volume will converge to the backbone network [2]. For alleviating this problem, information-centric networking (ICN) [3, 4] is proposed. And as a crucial branch of ICN, the Named Data Networking (NDN) [5–7] uses names instead of IP to rout and forward packets. Compared with the IP network, the requested content name is shifted from the application layer to the network layer to provide packet caching and hitting. Thanks to its support of content-centric in-network caching [8, 9], NDN is promising for application in large-scale content services [10–12]. Moreover, placing content caching at the edge network can also reduce network latency.

However, the caching capacity of edge devices is around 3 to 4 orders of magnitude lower than that in the cloud. Intuitively, the native caching policy is to cache every packet that arrived at nodes. Unfortunately, this implementation will result in very high content redundancy in the service network and waste many edge caching resources. Besides, the significant magnitude difference between the number of contents and edge node caching capacity will cause any caching eviction policy to become invalid. Because the following content will continuously replace the cached content in the node, and any valuable content cannot be kept in the node. Consequently, it is essential to distinguish popular content and reduce content redundant as much as possible in the edge caching process [13–15]. Moreover, since all of the in-network caching are triggered by user behaviors,

and the caching node can only passively cache arrived packets. So, the caching nodes cannot obtain explicit global knowledge about content popularity when making caching decisions. These all urge the adaptive caching decision mechanism to maximize caching utilization and reusability in resource-constrained edge networks.

For tackling the above problems, it is essential to design an in-network caching policy based on content popularity awareness. It should properly consider the characteristics of the user content requests and avoid additional overhead in the network. In addition, it is necessary to keep the caching redundancy as low as possible and achieve excellent robustness to adapt to various request scenarios. Given the above analysis, a popularity-aware in-network caching policy, namely, *Pop*, is proposed to achieve optimal in-network caching in the resource-constrained edge networks. Specifically, *Pop* allows the node to perceive the popularity of the current arrived content during the transmission process and then decide whether to cache based on the net location of the node. In the above process, there is no need to add additional hardware and traffic overhead.

Our contributions are listed as follows:

(1) We designed a popularity-aware in-network caching policy (*Pop*) for the edge network. It utilizes the existing Pending Interest Table (PIT) as a recording mechanism for current content request status. By distinguishing the request probability of content with different popularity, *Pop* naturally makes caching decisions and achieves distributed caching based on content popularity

(2) We introduced Cached Tag in the header of the returned packet to prevent the cached content from being cached again by downstream nodes, effectively reducing content redundancy in the edge network. Tag reset mechanism will be triggered by cache hit; it can effectively avoid popular content be incorrectly cached far from the edge and also can increase the utilization of edge caching space

(3) We conducted a comprehensive evaluation of *Pop* through network simulation. Compared with four existing policies and a self-designed ideal caching policy (*Ideal*), we prove that *Pop* has excellent performance. Besides, *Pop* also shows considerable advantages in terms of network-level caching distribution and content update responsiveness

(4) Complex and diverse network scenarios were simulated, which proves that *Pop* has good robustness. In large-scale and high-load scenarios, the performance of *Pop* is always better than other policies (except *Ideal*). In imbalanced-content request, *Pop* maintains its superior performance in most cases and only declines in a few extreme cases

The remainder of this paper is organized as follows. Section 2 introduces the related work of in-network caching. Section 3 introduces the system model and related assumptions. Section 4 introduces the details of the *Pop* in-network caching policy. In Section 5, performance evaluation of the *Pop* is showed. Section 6 discusses the variants of *Pop* and the real deployments in edge environments. Section 7 summarizes our work.

2. Related Work

Unlike traditional caching systems, the implementation of in-network caching can be divided into caching eviction process and caching decision process.

Caching eviction process is when the arrived content is determined to be cached, but the caching space is full, then it needs to decide which stale content should be evicted from the node. Well-known caching eviction policies include FIFO (First Input First Output), LFU (Least Frequently Used), Random, and LRU (Least Recently Used). Among them, LRU is the most widely used in ICN in-network caching research [16, 17]. In addition, LFF [18] (Least Fresh First) is used in IoT networks; it aims to use the ARMA model to predict the time of the next content update event in the sensor, so that the predicted remaining time will be set as the freshness of contents in caching nodes.

Caching decision process is when the content has arrived; the node needs to decide whether to cache it into caching space. Due to the limitation of node caching capacity, when all of the arrived contents are cached, the cache hit efficiency will be very low. As the native caching mechanism of the NDN, it allows every node to cache every arrived content, called LCE [6] (Leave Copy Everywhere). It is proven that results in high redundancy and low utilization in edge storage resources. Ber [19] (Bernoulli random caching) is a Probabilistic Caching policy that satisfies Bernoulli distribution, which allows content to be cached with a fixed probability on each node of the return path. Moreover, some researchers began to construct more complex formulas for optimizing caching probability. They introduced different parameters, which lead the caching probability to change dynamically according to node position and status. ProCache [20] (Probabilistic Caching) mainly introduces the distance and the remaining storage space, so that the higher caching probability will belong to the node close to the user and with more caching space. The pCASTING [21] (Probabilistic Caching strategy for the Internet of Things) policy applies to energy-constrained and wireless IoT networks. It introduces the remaining power, remaining storage space, and content freshness as caching probability parameters. It leads the power consumption of IoT equipment in a balanced way and effectively increases the working time of the entire IoT network. LCD [22] (Leave Copy Down) allows the returned (or cache hit) content to be cached only on its next-hop node, so the caching location of the content can be gradually moved to the user through multiple be requested. However, its upstream nodes will have many times of content eviction and redundancy in caching process.

On the other hand, other caching policies allow every content to be cached on only one node, so that to minimize network redundancy. The key to these policies is the selection of the caching node. The Ran (Random choice caching)

policy does not consider any external features and randomly selects one of the downstream nodes for caching. The Betw [23] (Betweenness centrality caching) policy believes that caching the content on the node with the largest betweenness centrality can maximize future benefits. Hash [24, 25] (Hash-routing caching) policy expands the range of caching nodes from the return path to the entire network. It uses a hash function to map content to nodes in different net locations by content names, but it has high implementation complexity.

Besides, there are other policies consider in other ways. For PPCS [26] (progressive popularity-aware caching scheme), it designs from caching decision and eviction. During the first transmission, PPCS divides the larger content into several smaller packets for distributed caching. If this content is requested multiple times in the following time, it can gradually push all packets of this content to the terminal caching nodes by an intelligent caching scheme. Newberry and Zhang [27] considered the in-network caching for the Hadoop distributed file system. By comparing multiple caching eviction policies, it is found that the size of the packet capacity has a more significant impact on the performance of the caching efficiency.

The above work has done much research on improving the efficiency of the in-network caching from many aspects. However, almost none of them considers the relationship between content popularity and its request probability in the tree-like network, nor did comparison with an ideal caching policy to clearly show the gap of performance between their policy and the ideal situation. Therefore, we propose the *Pop* policy and design *Ideal* policy to compare with it. We hope our work can bring a new view to the in-network caching research and help improve its performance.

3. System Model and Assumptions

We argue that the popularity of content is the crucial factor affecting the benefits of in-network. In Section 3.1, we describe a typical network architecture that we considered. In Section 3.2, we present the assumptions and characteristics of user requests.

3.1. System Model. Typical Internet content services are deployed in hierarchical network architectures, as shown in Figure 1. In fact, all devices on the path from the cloud to user can be used as caching nodes, but we focus on the nodes in the edge environment. Obviously, it can alleviate the pressure of the backbone network and get the low-latency response from edge caching nodes.

3.2. Request Assumptions. Internet content services model include diverse request scenario. We mainly consider the following characteristics and make some assumptions based on generality.

Request generation. We suppose the request generation meets the *Poisson* distribution.

Content Popularity. Due to the content preference of consumers, most requests tend to focus on a small amount of content. Relevant research [28] has observed that the popularity of the requested contents on the web meets *Zipf-like*

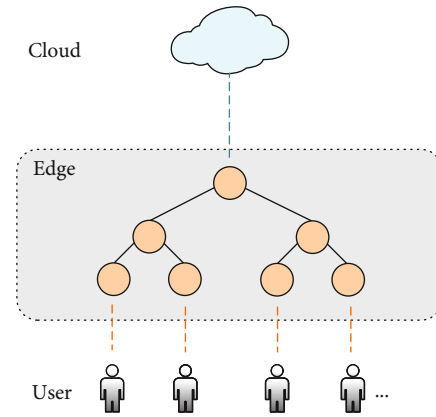


Figure 1: Hierarchical network architecture, where cloud node is the location of publishing server, and it interacts with users through edge network.

law. We suppose the content popularity distribution meets the *Zipf-Mandelbrot* law. When the total number of contents is N, the popularity ranking of the content is k; the probability that it be requested is

$$f(k, N, q, s) = \frac{1/(k+q)^s}{H_{N,q,s}}. \tag{1}$$

Here, q and s are parameters that affect the distribution, and $H_{N,q,s}$ is given as follows.

$$H_{N,q,s} = \sum_{i=1}^{N} \frac{1}{(i+q)^s}. \tag{2}$$

Request probability. In a tree-like network, requests will be collected in upstream nodes. When we consider the probability of a specific content has been requested over a period of time, its popularity will lead to very different probability distribution at every network level.

We consider the number of requests received by a node within a period of time is n, and the popularity of the content is p. Then, the probability P that this node received this content can be given as follows.

$$P = 1 - (1-p)^n. \tag{3}$$

The n is related to the user request frequency and the network topology, as shown in Table 1. If we suppose a standard z-branches tree network, the average frequency of user requests is x, and the total network level is K; the n of level k nodes can be given by (4)

$$n = z^{K-k}x. \tag{4}$$

A specific example is in Figure 2. It shows that contents with different popularity have different request probability distributions at caching nodes. Take $P = 0.9$ (thin dotted line in the figure) as a reference. 3.00% popularity content from cloud to $L4$ can receive its request more than 90%

Table 1: 6-layer standard tree network request distribution.

Level	Number of received requests n		
	Binary tree	Trinomial tree	z-branches tree
$L1$	$32x$	$243x$	z^5x
$L2$	$16x$	$81x$	z^4x
$L3$	$8x$	$27x$	z^3x
$L4$	$4x$	$9x$	z^2x
$L5$	$2x$	$3x$	zx
$L6$	x	x	x

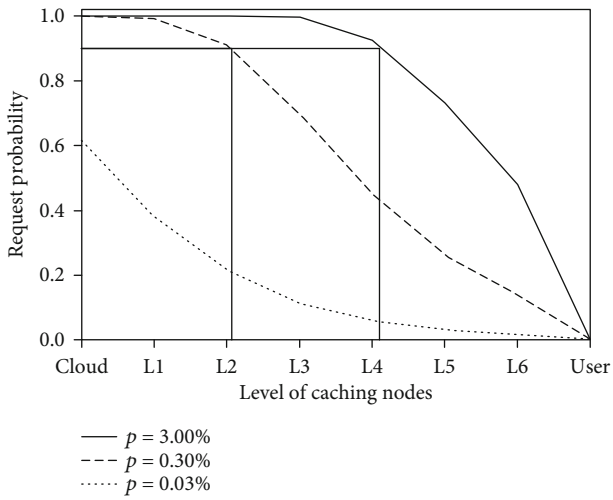


Figure 2: An example about the request probability distribution of content with different popularity in the binary tree network. The popularity of the three contents is 3.00%, 0.30%, and 0.03%; the frequency of single-user requests is 50. There is no caching mechanism in the user node, so the probability is 0 in user.

probability. In comparison, content with 0.30% popularity from cloud to $L2$ can receive its request more than 90% probability. As for content with a popularity of 0.03%, the request probability has always been lower than 62%.

4. Popularity-Aware In-Network Caching Policy

Identifying the popularity of contents and caching them to the suitable network level is crucial for the popularity-aware caching policy (*Pop*). This section will present a demonstration of *Pop* first and then introduce the implementation of the key mechanisms in detail.

According to the analysis in Section 3.2, the popularity of content will affect the request probability distribution in each level node. So, we can distinguish the popularity of contents according to the request records in caching nodes.

As a demonstration, we still use the standard binary tree network to explain main idea of the *Pop* policy. We use LRU as the eviction policy due to its excellent performance (evaluation details in Section 5.2), and the processing of caching decision is showed as Figure 3. Now, we consider a user node sends three contents requests to the cloud. They are high popularity content *Ra*, medium popularity content *Rb*, and low popularity content *Rc*. These PIT tables (NDN module, details in Section 4.1) will record three content requests (*Ra*, *Rb*, and *Rc*), and their in-records are all marked in the right sub-Face. After that, while waiting for the cloud content return, the *N1*, *N2*, and *N3* nodes will continue to receive content requests (*R* shown in Figure 3) from the upper sub-Face. According to the analysis of request probability, content popularity, and node location in Section 3.2. The *N1* node has a great chance to receive *Ra* and *Rb* content requests on the upper sub-Face. *N2* node may receive *Ra* request. The probability of the *N3* node receiving these content requests is extremely low.

Therefore, when *Ra*, *Rb*, and *Rc* are returned, each caching node will make the caching decision based on the PIT table. We set the caching condition that the returned content will be cached in the first node, which does not record its request in both sub-Faces. According to the records shown in Figure 3, node *N1* will cache *Rc*, node *N2* will cache *Rb*, and node *N3* will cache *Ra*. So far, the *Pop* policy has completed a distributed caching based on the content popularity and caching node network level.

Besides, the mechanism for reducing caching redundancy is in Section 4.2. The detail and the related algorithm extend the above caching condition to a more generalization and complex hierarchical network are in Section 4.3.

4.1. Request Recording Mechanism. We use NDN as the underlying network. NDN allows establishing communication interfaces between nodes through various underlying protocols, such as Ethernet, TCP, UDP, and Socket. They are unified and abstracted as Face. The communication of NDN is launched by the users, and their requests will be encapsulated in a packet called *Interest* and routed to the cloud. The content returned by the cloud will be encapsulated in a packet called *Data* and routed to users. The PIT (Pending Interest Table) is the native mechanism of NDN, whose function is to record the name of *Interest* that the node has forwarded but has no *Data* returned from the upstream. Therefore, we can directly use PIT as the request recording mechanism and avoiding the additional design.

As shown in Table 2, PIT mainly consists of three parts. Entry is the Interest name that has been forwarded. In-record is the downstream source Face ID. Out-record is the upstream forwarding Face ID. The records in the table indicate that the node has received *Ra* and *Rb* requests, where *Ra* has been requested on both Faces 1 and 2, and *Rb* only has requested on Face 2. Do not consider other records. According to the above caching conditions, *Ra* will be directly forwarded to Faces 1 and 2; *Rb* will be cached and then forwarded to Face 2. And then, the node will immediately clear all *Ra* and *Rb* records in the PIT. The maintenance of the PIT table is implemented by the Named Data Networking Forwarding Daemon (NFD) [29].

4.2. Cached Tag Design of the Returned Data. Only based on the above caching condition, it cannot avoid the returned

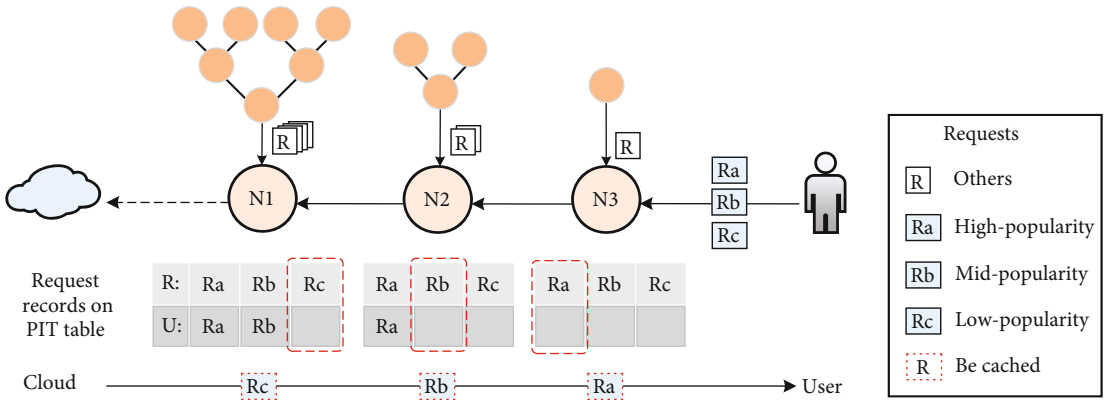


Figure 3: Demonstration of *Pop* in distinguishing the popularity of contents and caching them in opportune node. The user sends different popularity requests to the cloud. When contents are returned, they are cached based on their popularity and caching node network level.

Table 2: Record of received Interest in PIT.

Entry	In-record	Out-record
/prefix/Ra	Face 1	Face 0
/prefix/Ra	Face 2	Face 0
/prefix/Rb	Face 2	Face 0

Data be repeated caching on the return path. We designed the Cached Tag to solve this problem.

As shown in Figure 4, we denote the returned *Data* as *C* and add the Cached Tag to its header field. When *Data C* returns from cloud, its Tag is initialized to 0. As the *Data C* hops to the user node along the return path, it will go through the Tag detection process on each caching node. At node *N1*, *Data C* does not meet the caching condition, the value of Tag unchanged, and it is directly forwarded to node *N2*. At node *N2*, *Data C* meets the caching condition; the value of Tag is changed to 1, and then, it is forwarded to node *N3* after caching. After receiving *Data C*, the *N3* node checks that the Tag of *C* is 1, so it directly abandons the caching and forwards it to the user.

Besides, we also designed a Tag reset mechanism to optimize the efficiency of content caching further. Specifically, when the request triggers cache hits on the caching node, the returned *Data* is allowed to reset the Tag value to 0, thereby obtaining another caching opportunity. The Tag reset mechanism will generate benefits from two aspects.

First, solve the problem of content caching mismatch. The caching decision of *Pop* cannot be guaranteed 100% accuracy. It may occur that the cached content and the cached location do not match. As shown in Figure 5, a mismatch content (with high popularity) is incorrectly cached on a far node. Here, the Tag reset mechanism allows mismatch content to be cached at the next cache hit. The popularity-aware mechanism will smoothly cache it to the matching node (edge node).

Second, use the invalid (stale or idle) caching resources of edge nodes and optimize cache hit efficiency. The distribution of content popularity is imbalanced. The amount of high or medium popularity content is far less than the low. Reflected in the caching results, strictly caching based on content popularity will lead to a low caching utilization in middle or edge nodes. Here, the reset mechanism allows these contents on the far nodes to be cached to the middle or edge nodes, reducing the number of hops for following repeat requests.

The Tag reset mechanism will not impact the valid content in the caching nodes. The LRU caching eviction policy is deployed in each node, and it will maintain the currently popular content at the top.

4.3. Threshold-Based Adaptive Caching Decision Algorithm. Given the complexity and heterogeneity of the real network, *Pop* policy must adaptively complete the caching decision process.

As shown in Figure 6, we introduce a threshold *T* to implement the *Pop* adaptive caching decision in the diversified nodes of the nonstandard tree network. We abstracted *Request_ratio*, which represents the ratio between *Ncur* (the number of sub-Faces with this content request) and *N* (the total number of sub-Faces). *Request_ratio* describes the popular degree of content on downstream nodes, and the value range is (0, 1]. The higher the value, the more popular the content.

In Figure 6, the threshold *T* is set to 0.6. The (a) depicts a 3-branch caching node with three sub-Faces, and the (b) depicts a 5-branch caching node with five sub-Faces. Based on the unified threshold *T*, they can make the same caching decisions on *Ra* and *Rb*.

Algorithm 1 is deployed on all caching nodes uniformly. The input includes returned *Data*, *PIT*, number of sub-Faces *N*, and the custom caching threshold *T*. When *Data* returns from the upper Face of the node, the caching decision algorithm is triggered. The algorithm first checks the *CacheTag* in the *Data* (line 2). If $ChaheTag! = 0$, it proves the upstream nodes have cached the *Data*, forward the *Data* directly (line 2), and the process finishes. Otherwise, *Data* will enter the caching decision process (lines 5-15).

For caching decision process, the algorithm decodes the *Data* and obtains the content name. It then queries the *PIT* according to the name and gets the *N_cur* (line 6). Next, calculate *Request_ratio* (line 7). By comparing *Request_ratio* and threshold *T*, the algorithm can obtain two decision

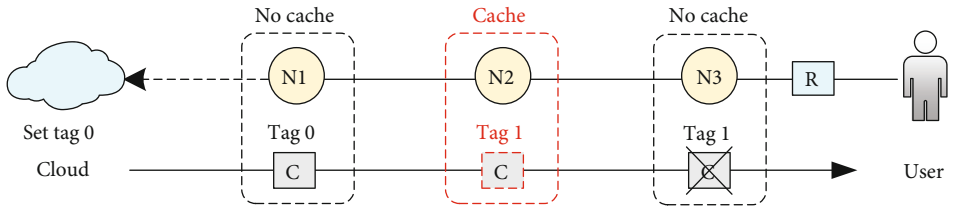


Figure 4: The process of returned *Data* with Cached Tag. The return *Data* with initial Cached Tag 0, no cache in *N1*, change Tag to 1 after caching in *N2*, give up cache in *N3*.

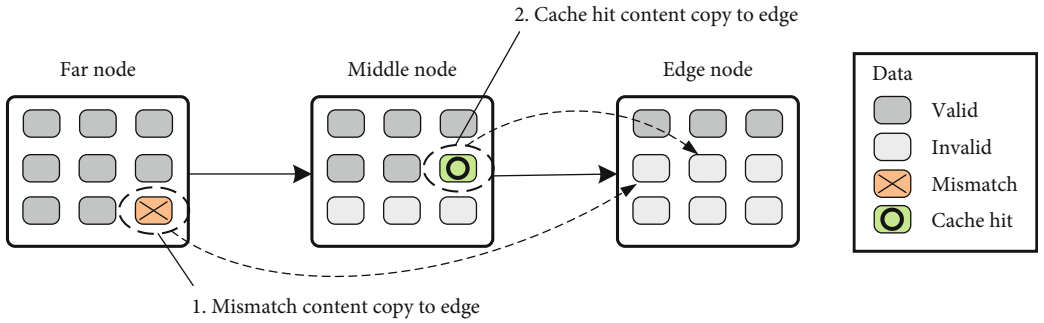


Figure 5: Two cases of Cached Tag reset. The mismatch content and cache hit content will have additional chance to cache close to edge.

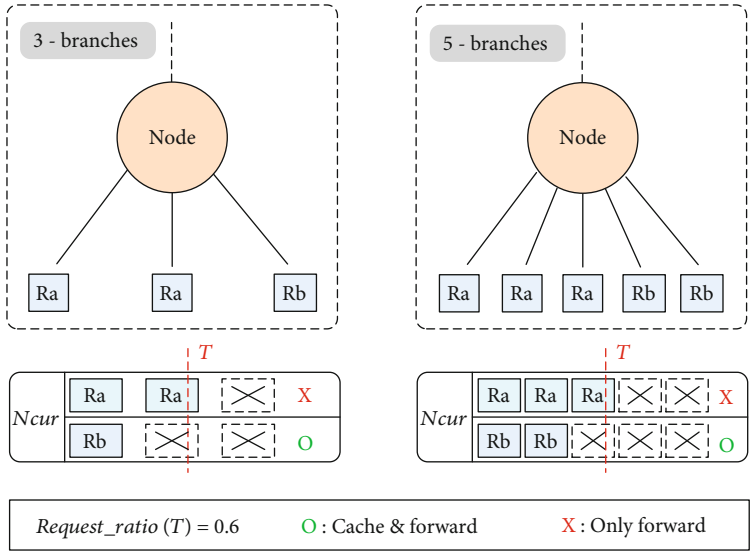


Figure 6: Adaptive content caching based on *Request_ratio*. Set $T = 0.6$. In (a), *Ra* is not cached ($2/3 > T$), and *Rb* is cached ($1/3 < T$). In (b), *Ra* is not cached ($3/5 > T$), and *Rb* is cached ($2/5 < T$).

results. If the *Request_ratio* is greater or equal to T (line 8), it proves that the content popularity is too high and does not match this caching node, and the *Data* will be forwarded directly (line 10). Otherwise, proving that the content popularity meets the current node caching requirement. The algorithm first sets the *Data.CacheTag* to 1 (line 13), then cache it (line 14), and finally forwards it (line 15) to the sub-Faces.

5. Performance Evaluation

In order to verify the effectiveness of *Pop*, we used ndnSIM [30–33] to perform a comprehensive simulation. Section 5.1 introduces the parameter configuration and indicators. Section 5.2 is the determination of the eviction policy, and Section 5.3-5 analyzes the performance in different request scenarios.

5.1. Simulation Design. The parameter configuration of ndnSIM is shown in Table 3. Please note that some parameters will change in follow simulation scenarios, and their change range is marked with "{...}."

For the evaluation of in-network cache policy, the main performance metrics are designed as follows.

```
Input: Data - Packet of Data;
       PIT - PIT table of NDN;
       N - Number of sub-Face;
       T - Threshold for caching content;
Output: Caching Decisions - (i). Cache & Forward; (ii). Only Forward.
//Deployed in all of caching nodes
1.  While ( new Data is return ) {
2.    if (Data.CacheTag != 0 ) {
3.      Forward(Data) // This Data has cached in upstream node, just forward it
4.    }else{
5.      // N_cur is number of sub-Face which has requested this Data
6.      N_cur = Search(http://Data.name, PIT)
7.      Request_ratio = N_cur / N // Get Request_ratio
8.      if (Request_ratio >= T){
9.      // Data was requested on most sub-Face, it's suitable for caching at downstream node
10.       Forward(Data) // Forward Data
11.   }else{
12.     // Data was requested on few sub-Face, it's suitable for caching at here
13.     Data.SetCachedTag(1) // Set cached Tag = 1
14.     Cache(Data) // Cache Data
15.     Forward(Data) // Forward Data to downstream nodes
16.   }
17. }
18. }
```

Algorithm 1: *Pop* caching decision algorithm.

Table 3: Simulation parameters.

Type	Name of parameter	Value
Network	Basic network topology	Binary tree
	Network level	5-level
	Link capacity to cloud	1 Gbps
	Link capacity in edge	100 Mbps
	Propagation delay to cloud	500 ms
	Propagation delay in edge	1 ms
Publish	Total number of contents	100,000 {20,000~ 100,000}
	Content update cycle	5 s
Request	Number of user nodes	16 {16~ 81}
	Request frequency	200/s {100~ 300}
	Request generation	*Poisson*
	Content popularity	*Zipf-Mandelbrot*
	Zipf parameters (q&s)	1.0 and 1.0
Caching	Caching eviction strategy	LRU {Random, LFU, FIFO, TTL}
	Caching node capacity	50
	Payload size (content size)	1024 Byte
	Pop caching threshold T	0.6
Simulation	Simulation duration time	201 s

(1) *Cloud Service Hit Reduction Ratio*. This indicator measures the in-network caching policy efficiency in reducing the pressure of the cloud. We use H_C as the number of nonfirst requests cache hit on cloud node and H_E as the number of requests cache hit on edge caching nodes. The value range of α is (0,1)

$$\alpha = 1 - \frac{H_C}{H_C + H_E} = \frac{H_E}{H_C + H_E}. \quad (5)$$

(2) *User Request Ave-Hop Reduction Ratio*. This indicator measures the efficiency of in-network caching

policies in improving the quality of user requests. *Ave_hops* represent the average hops to complete the user requests under caching policies, and *Hops_to_Cloud* represents the total hops from the user to the cloud. The value range of β is (0, 1/*Hops_to_Cloud*)

$$\beta = 1 - \frac{\text{Ave\_Hops}}{\text{Hops\_to\_Cloud}}. \quad (6)$$

(3) *Single-Layer Contribution*. This indicator measures each layer caching nodes' contribution in reducing the pressure of the cloud. It can also be understood as the probability of triggering cache hit on each layer. Where l represents the level of nodes, H_{El} represents the total number of requests cache hit on layer l. $\sum H_{El}$ is the total number of requests cache hit on the entire edge network

$$\gamma(l) = \frac{H_{El}}{\sum H_{El}}, l = 1, 2, \cdots \cdots, n. \quad (7)$$

(4) *Ideal In-Network Caching Policy*. To test the *Pop* accuracy in perceiving content popularity, we design the *Ideal* policy as a comparison. In its implementation, the *Data* returned by the cloud will carry the content popularity information. It allows the caching nodes to make caching decisions based on the known popularity information and achieve the ideal caching performance. Through comparison, we can see the gap between the *Pop* and the *Ideal*

5.2. Determination of the Eviction Policy. Before the *Pop* performance evaluation, we need to determine the caching eviction policy. Under the above default configuration, we have performed the evaluation in four policies, included Random, LFU, FIFO, TTL, and LRU. As shown in Figure 7, Random and LFU all show the low reduction ratio; FIFO and TTL have similar performance, and LRU has the best performance.

Therefore, based on the above comparison results, it can find that the LRU caching eviction policy has appropriate performance. So, in the following experimental scenarios, *Pop* and other comparison policies will use LRU as the caching eviction policy.

5.3. Content Update. We thoroughly evaluated the performance of the *Pop* in content update scenarios. By comparing with other existing in-network caching policies (LCE, Probcache, LCD, Ran) and *Ideal* policy, we prove the significant advantages of *Pop*.

(1) *In-Network Caching Benefit.* As shown in Figure 8, excepts for the *Ideal* policy, *Pop* has the best performance in the reduction ratio of cloud service hit and user request average hops

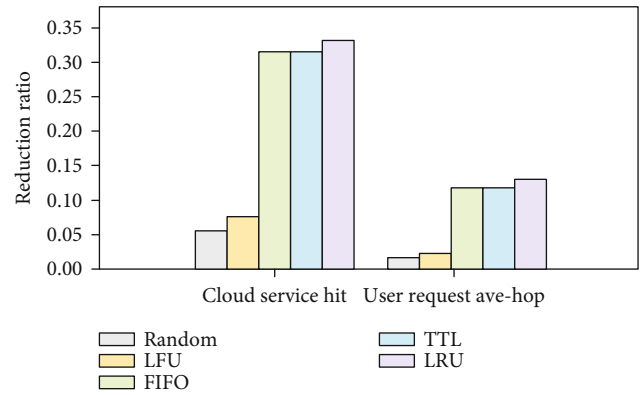


Figure 7: Comparison of cloud service hit and user request ave-hop reduction ratio in eviction policies. LRU has the best performance; its reduction ratio is 33.07% and 12.97%, respectively.

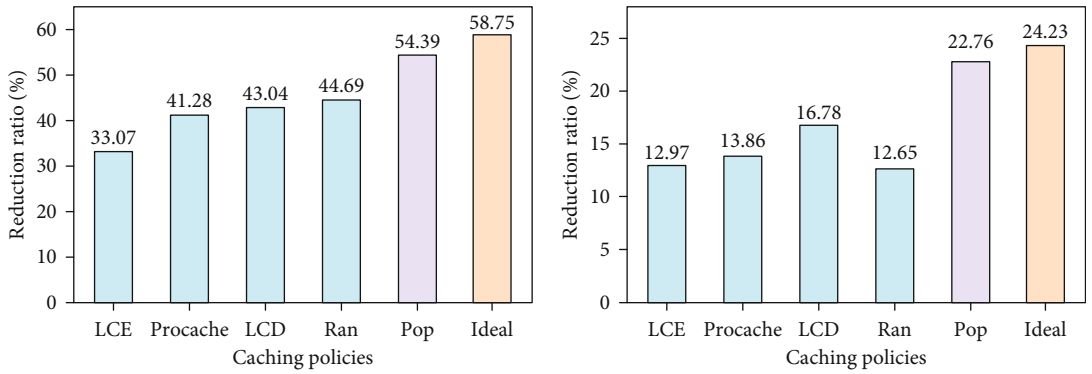
As Figure 8(a), the *Pop* reduced 54.39% of cloud service cache hit. Compared with the native policy LCE, the performance is improved by 21.32%. Compared with the suboptimal policy Ran, the performance is also improved by 9.7%. The performance gap between *Pop* and *Ideal* is 4.36%. As Figure 8(b), *Pop* has reduced 22.76% of user request average hops. Compared with other existing policies, the improvement range is 5.98% ~ 10.11%, and the gap with *Ideal* was only 1.47%.

It is worth noting that, due to the random selection of caching nodes, the Ran has well performance in the reduction ratio of cloud service hit, but its performance on reduction of average user request hops is lower than all of the other policies.

(2) *Contribution of Each Layer*. Furthermore, we counted the contribution of each layer. Figure 9 shows the distribution of the single-layer contribution in different in-network caching policies. Except for Ran, the contribution distribution of other caching policies decreases as the network level close to cloud ($L1$). The *Pop* caching result is very close to *Ideal*, which can reflect that *Pop* popularity awareness is accurate. Ran randomly selects the caching nodes so that all of the content is evenly cached on each layer nodes

(3) *Responsiveness to Content Updates*. Content updates will make the cached contents stale and invalid; they should be replaced by new contents quickly. In order to evaluate the *Pop* performance under dynamic scenarios, we show the responsiveness of different in-network caching policies

As shown in Figure 10, it fully proves that *Pop* responds very quickly to content updates (except for the *Ideal*). Specifically, the average hops of each caching policy change periodically. After every content update, the *Ideal* finishes new content caching at first, thereby quickly reducing the average hops. The *Pop* policy is second only to the *Ideal*. In the initial



(a) Cloud service hit reduction ratio

(b) User request ave-hop reduction ratio

Figure 8: Comparison of cloud service hit and user request ave-hop reduction ratio in different in-network caching policies. *Pop* reduced 54.39% of cloud service cache hit and 22.76% of user request ave-hop.

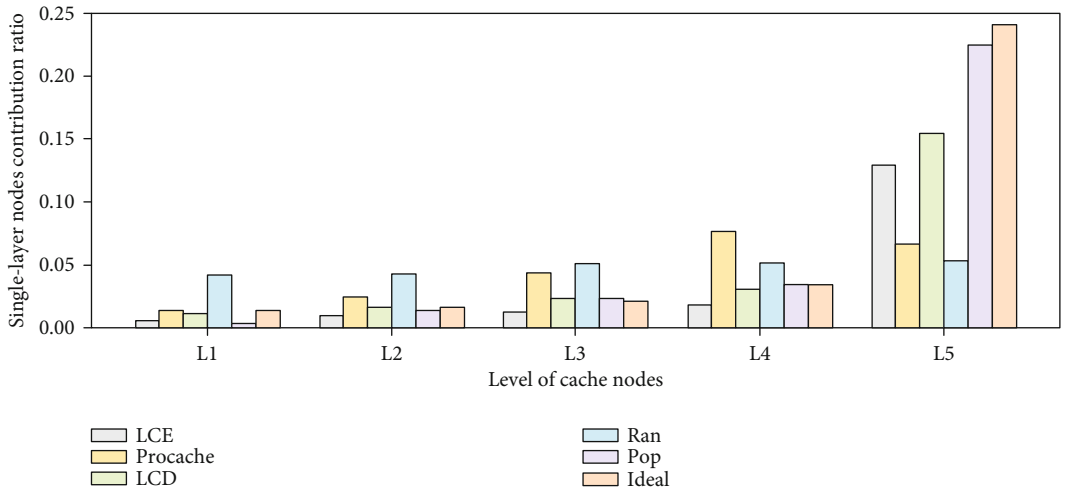


Figure 9: Comparison of single-layer contribution in different in-network caching policies. *Pop* caching result was close to *Ideal*.

stage, the *Pop* can quickly reduce the average hops at almost the same speed as *Ideal*. However, in the following stages, the reduction of the *Pop* was slightly insufficient. It shows that *Pop* can accurately perceive most high or medium popularity content and quickly cache them directly to the appropriate nodes. However, for some low popularity content, its perception ability may not be sensitive.

5.4. Large-Scale and High-Load. To test the *Pop* policy robustness, we changed different simulation parameters based on the above content update scenario. It can show the *Pop* performance from three dimensions: the number of published contents, the scale of the edge user, and the frequency of user requests.

As shown in Figure 11(a), when the number of published contents gradually increases (from 20,000 to 100,000), the cache hit reduction ratio of LCE, LCD, Procache, and Ran policies all dropped slightly. Only the *Pop* and *Ideal* policies showed an upward trend. It proves that facing the explosive growth of content volume, the in-network caching policy based on popularity-aware has significant advantages.

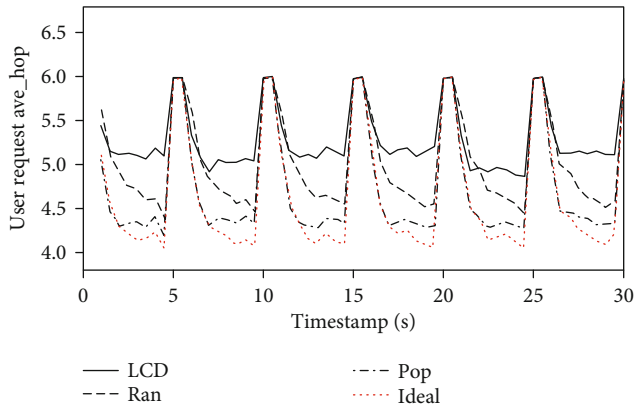


Figure 10: Comparison of content update responsiveness in different in-network caching policies. Content update cycle is 5 s; statistics period of average hops is 500 ms. Due to the delay, the line does not start at time 0, and there is a short interval after each update.

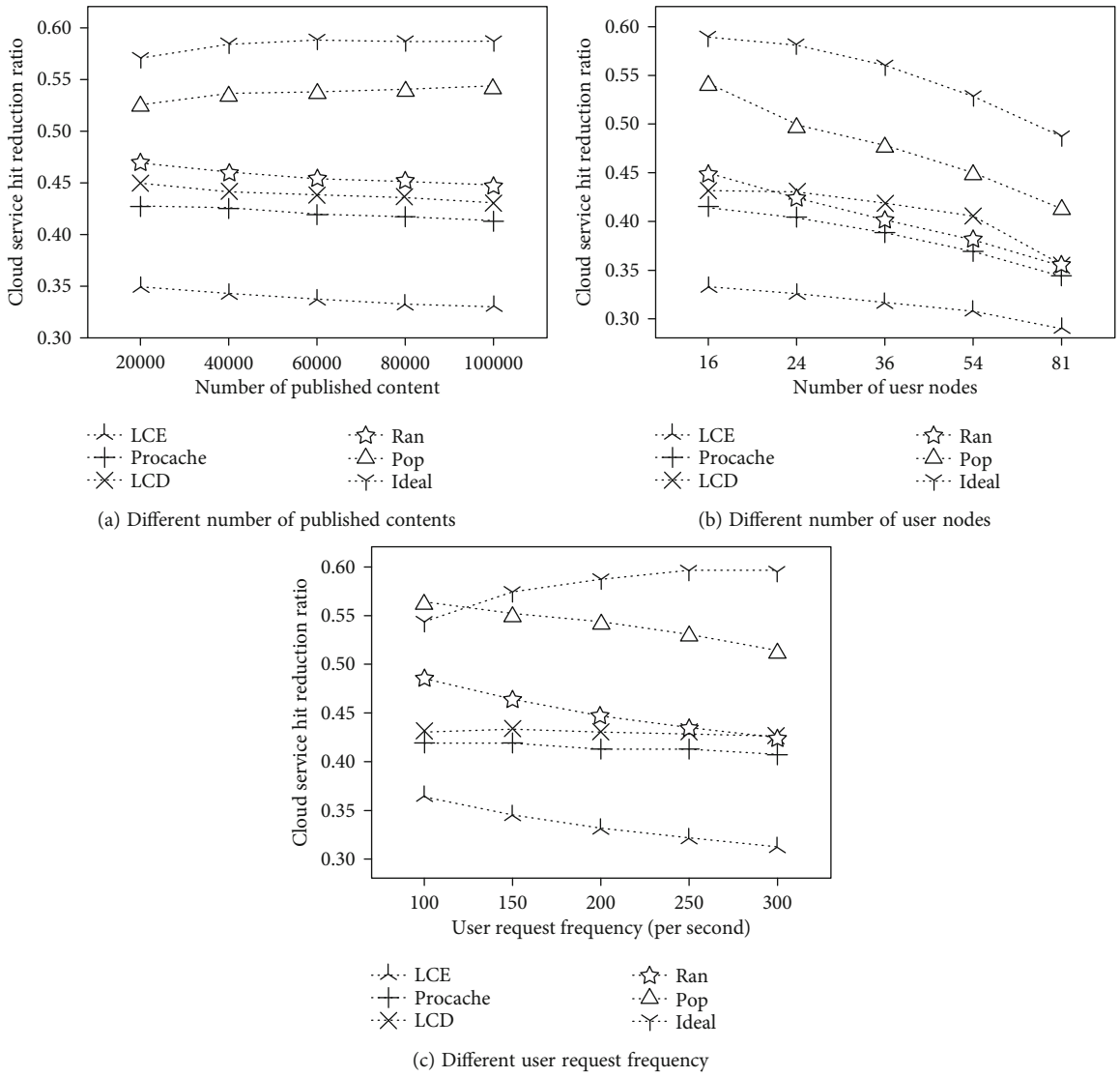


Figure 11: Cloud service hit reduction ratio in large-scale and high-load scenarios.

Figure 11(b) describes the cache hit reduction ratio in the different number of user nodes. All caching policies have the downward trend. However, within this experiment range, the *Pop* cache hit reduction ratio remains between 42% and 55%. Compared with other policies (except *Ideal*), it still has the best performance.

Figure 11(c) is an evaluation of different user request frequencies. Except the *Ideal* policy, the cache hit reduction ratio of other caching policies all meet the downward trend. Besides, it is worth noting that *Pop* performance is better than *Ideal* when the request frequency is 100.

The above three sets of simulation experiments verify the *Pop* excellent performance and robustness in large-scale and high-load content update scenarios. It also shows the considerable potential of deploying *Pop* in real networks in the future.

5.5. Imbalanced-Content Request. We considered the imbalanced-content request scenario to test the *Pop* performance degradation under extreme conditions.

In the experimental design, we divided the imbalance degree of user request into 16 levels, represented as $L0 \sim L15$. We still use the default network topology and parameter configuration, but some adjustments have been made to the published contents and user node request process. Specifically, the 100,000 published contents are divided into 16 groups on average. Each group has 6,250 contents and has an independent popularity distribution. Correspondingly, we also created 16 different request processes on each user node, and each process is only allowed to request one group of contents. When the imbalance level is $L0$, each user node starts 16 request processes, and the requested content pool is 100,000. When the content request level is $L15$, each user node can only start one content request process, and the requested content pool capacity is 6250. The requested content pool capacity is no overlapping part.

As shown in Figure 12, for the convenience of observation, the cache hit reduction ratio is normalized to *Ideal* policy. We can find that the *Pop* policy can still guarantee a high

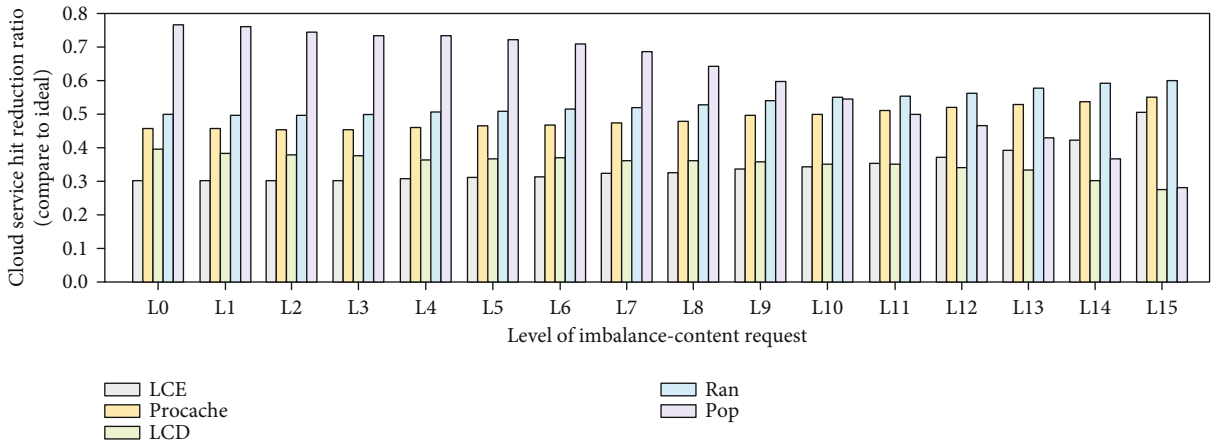


Figure 12: Cloud service hit reduction ratio (compare to *Ideal*) in imbalanced-content request.

reduction ratio in most imbalanced levels. Specifically, the *Pop* policy has the best performance at the $L0 \sim L9$ levels. At $L10 \sim L13$ levels, the hit reduction ratio remains in the top three. Nevertheless, under the extreme conditions of $L14$ and $L15$, its performance drops rapidly.

In the real content request scenario, requests for high popularity content have commonalities. It generally does not happen that all users have an independent content interest. The above evaluation results show that *Pop* can be applied to most request cases.

6. Discussion

This section will discuss the content that was considered during the research but not described above.

6.1. Variant. Under the *Pop* policy, high popularity content will be forwarded to downstream nodes for caching, and low popularity content will be cached on upstream nodes. However, in a tree network, the number of upstream nodes is much smaller than that of downstream nodes. In the above mechanism, similar high popularity content will be cached in many downstream nodes. It will increase the redundancy of the edge network. An opposite variant is to cache the high popularity contents on the upstream caching nodes and store low popularity content in downstream nodes. This variant can effectively avoid the redundancy of high popularity contents, but the requests will be satisfied on the farther caching node.

Moreover, caching high popularity contents at the upstream caching nodes will making them to be another "network center." A large number of requests will be converged to the upstream nodes, and the bandwidth bottleneck that existed in the cloud will be transferred to these nodes. It goes against the original intention of in-network caching that distributes the cloud pressure to the entire network to achieve more efficient resource utilization and service response.

In fact, the in-network caching is essentially a technology that uses "space" to exchange "efficiency." The deployment of caching policy is inevitable to increase the content redundancy of the network. In the design of caching policy, we should focus on reducing the redundancy of the request path, not the redundancy of the entire network.

6.2. Real Deployment. Given much research on the IoT and edge computing [34–40], introducing containerization and serverless platform to the edge environment maybe is the best practice to implement *Pop*.

First, due to the compatibility between NDN and IP network, *Pop* can be easily deployed on traditional edge servers. Secondly, the number of requests will change over time; it requires that the resources used for caching be dynamically adjusted according to the actual workload. By using the container-based serverless platform can solve this problem very well. It can automatically control the number of containers and effectively weighing the balance between service quality and resource utilization. Finally, in the design of *Pop*, the information used in the caching decision algorithm can be obtained by itself and does not need to communicate with other external nodes. Therefore, it is possible to use stateless functions to implement *Pop* and dramatically simplifies its deployment.

In addition, the caching capacity of caching nodes and the threshold of caching decisions can be considered dynamically adjusted according to the current workload. In this way, load balancing and caching efficiency of the entire edge network node may be achieved.

7. Conclusions

We propose a popularity-aware in-network caching policy (*Pop*). The design of *Pop* makes full use of the PIT table and the distribution of request probability in the tree-like network. At the same time, the design of the Cached Tag in the *Data* header ensures low content redundancy in the edge network, and the Tag reset mechanism effectively optimizes the performance of *Pop* caching.

For the further work, we will comprehensively evaluate the performance of *Pop* and prove that it has obvious advantages over other existing in-network caching policies in many aspects. In addition, through the design of multiple scenarios, we have proven that the *Pop* policy has good

robustness. We also discussed the significance of in-network caching and the possibility of introducing containerization and serverless platform to the implementation of *Pop*.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest in the work.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (No. 61972118) and the Key Research and Development Program of Zhejiang Province of China (No. 2019C01059).

References

[1] N. Fotiou, D. Trossen, and G. C. Polyzos, "Illustrating a publish-subscribe Internet architecture," *Telecommunication Systems*, vol. 51, no. 4, pp. 233–245, 2012.

[2] F. J. M. Muro, N. Skorin-Kapov, and P. Pavon-Marino, "Revisiting core traffic growth in the presence of expanding CDNs," *Computer Networks*, vol. 154, pp. 1–11, 2019.

[3] X. Qiao, H. Wang, W. Tan, A. V. Vasilakos, J. Chen, and M. B. Blake, "A survey of applications research on content-centric networking," *China Communications*, vol. 16, no. 9, pp. 122–140, 2019.

[4] D. Mars, S. Mettali Gammar, A. Lahmadi, and L. Azouz Saidane, "Using information centric networking in Internet of Things: a survey," *Wireless Personal Communications*, vol. 105, no. 1, pp. 87–103, 2019.

[5] L. Zhang, D. Estrin, J. Burke et al., "Named data networking (NDN) project," Dept. Comput. Sci., Univ. California, Los Angeles, CA, USA, 2010, Tech. Rep. NDN-001.

[6] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pp. 1–12, Rome, Italy, 2009.

[7] N. D. N. Project, "Named data networking," February 2021, http://named-data.net/.

[8] M. H. Al-Adhaileh, F. Muchtar, A. H. Abdullah, and P. K. Singh, "The Significance of Using NDN in MANET," *Proceedings of ICRIC*, , pp. 421–437, Springer, Cham, Switzerland, 2020.

[9] Z. Yan, Y. Park, Y. Leau, L. Ren-Ting, and R. Hassan, "Hybrid network mobility support in named data networking," in *2020 International Conference on Information Networking (ICOIN)*, pp. 16–19, Barcelona, Spain, 2020.

[10] D. Gupta, S. Rani, S. H. Ahmed, and R. Hussain, "Caching Policies in NDN-IoT Architecture," in *Integration of WSN and IoT for Smart Cities*, pp. 43–64, Springer, Cham, Switzerland, 2020.

[11] M. Ehsanpour, S. Bayat, and A. M. A. Hemmatyar, "An efficient and social-aware distributed in-network caching scheme in named data networks using matching theory," *Computer Networks*, vol. 158, pp. 175–183, 2019.

[12] H. Gao, X. Qin, R. J. D. Barroso, W. Hussain, Y. Xu, and Y. Yin, "Collaborative learning-based industrial IoT API recommendation for software-defined devices: the implicit knowledge discovery perspective," *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 1–11, 2020.

[13] S. Lee, I. Yeom, and D. Kim, "T-caching: enhancing feasibility of in-network caching in ICN," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1486–1498, 2020.

[14] Y. Liu, T. Zhi, H. Xi, W. Quan, and H. Zhang, "A novel cache replacement scheme against cache pollution attack in content-centric networks," in *2019 IEEE/CIC International Conference on Communications in China (ICCC)*, pp. 207–212, Changchun, China, 2019.

[15] Y. Meng, M. A. Naeem, R. Ali, and B. S. Kim, "EHCP: an efficient hybrid content placement strategy in named data network caching," *IEEE Access*, vol. 7, pp. 155601–155611, 2019.

[16] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian et al., "Less pain, most of the gain," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 147–158, 2013.

[17] A. Sharma, A. Venkataramani, and R. K. Sitaraman, "Distributing content simplifies ISP traffic engineering," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 229–242, 2013.

[18] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, K. Drira, and H. Mathkour, "Least fresh first cache replacement policy for NDN-based IoT networks," *Pervasive and Mobile Computing*, vol. 52, pp. 60–70, 2019.

[19] L. Saino, I. Psaras, and G. Pavlou, "Icarus: a caching simulator for information centric networking (ICN)," in *7th International ICST Conference on Simulation Tools and Techniques*, pp. 66–75, Lisbon, Portugal, 2014.

[20] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pp. 55–60, Helsinki, Finland, 2012.

[21] M. A. Hail, M. Amadeo, A. Molinaro, and S. Fischer, "Caching in named data networking for the wireless Internet of Things," in *2015 international conference on recent advances in internet of things (RIoT)*, pp. 1–6, Singapore, 2015.

[22] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD interconnection of LRU caches and its analysis," *Performance Evaluation*, vol. 63, no. 7, pp. 609–634, 2006.

[23] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache 'less for more' in information-centric networks," in *International Conference on Research in Networking*, pp. 27–40, Springer, Berlin, Germany, 2012.

[24] L. Saino, I. Psaras, and G. Pavlou, "Hash-routing schemes for information centric networking," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pp. 27–32, Hong Kong, China, 2013.

[25] X. Hu, S. Zheng, L. Zhao, G. Cheng, and J. Gong, "Exploration and exploitation of off-path cached content in network coding enabled named data networking," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pp. 1–6, Chicago, IL, USA, 2019.

[26] Q. N. Nguyen, J. Liu, Z. Pan et al., "PPCS: a progressive popularity-aware caching scheme for edge-based cache redundancy avoidance in information-centric networks," *Sensors*, vol. 19, no. 3, p. 694, 2019.

[27] E. Newberry and B. Zhang, "On the power of in-network caching in the Hadoop distributed file system," in *Proceedings of the 6th ACM Conference on Information-Centric Networking*, pp. 89–99, Macao, China, 2019.

[28] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications," in *IEEE INFOCOM'99 Conference on Computer Communications*, pp. 126–134, New York, NY, USA, 1999.

[29] A. Afanasyev, J. Shi, B. Zhang et al., "NFD developer's guide," Dept. Comput. Sci., Univ. California, Los Angeles, CA, USA, 2014, Tech. Rep. NDN-0021.

[30] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," Dept. Comput. Sci., Univ. California, Los Angeles, CA, USA, 2012, Tech. Rep. NDN-0005.

[31] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2.0: a new version of the NDN simulator for NS-3," Dept. Comput. Sci., Univ. California, Los Angeles, CA, USA, 2015, Technical Report NDN-0028.

[32] NS-3 based Named Data Networking (NDN) simulatorFebruary 2020, https://ndnsim.net/current/index.html.

[33] NS-3 Network SimulatorFebruary 2020, https://www.nsnam.org/.

[34] A. M. Aske, "Scheduling functions-as-a-service at the edge," Ph.D. dissertation, Dept. Computer, Washington State University, Pullman, WA, USA, 2018.

[35] L. Baresi and D. F. Mendonça, "Towards a serverless platform for edge computing," in *2019 IEEE International Conference on Fog Computing (ICFC)*, pp. 1–10, Prague, Czech Republic, 2019.

[36] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, H. Karimipour, G. Srivastava, and M. Aledhari, "Enabling drones in the Internet of Things with decentralized blockchain-based security," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6406–6415, 2021.

[37] Y. Liu, J. Wang, J. Li et al., "Zero-bias deep learning for accurate identification of Internet-of-Things (IoT) devices," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2627–2634, 2021.

[38] Y. Wu, "Cloud-edge orchestration for the Internet of Things: architecture and AI-powered data processing," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12792–12805, 2021.

[39] X. Li and L. D. Xu, "A review of Internet of Things—resource allocation," *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 8657–8666, 2021.

[40] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Toward collaborative inferencing of deep neural networks on Internet-of-Things devices," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4950–4960, 2020.