Cache-Filter: A Cache Permission Policy for Information-Centric Networking

Bohao Feng[1], Huachun Zhou[1], Mingchuan Zhang[1,2], Hongke Zhang[1]
[1]Institute of Electronic and Information Engineering, Beijing Jiaotong University
Beijing, Beijing 100044 - China
[e-mail: {bohaofeng, hchzhou, zhang_mch, hkzhang}@bjtu.edu.cn]
[2]Information Engineering College, Henan University of Science and Technology
Luoyang, Henan 471023 - China
*Corresponding author: Bohao Feng

Received June 12, 2015; revised August 24, 2015; revised September 16, 2015; accepted September 28, 2015; published December 31, 2015

Abstract

Information Centric Networking (ICN) has recently attracted great attention. It names the content decoupling from the location and introduces network caching, making the content to be cached anywhere within the network. The benefits of such design are obvious, however, many challenges still need to be solved. Among them, the local caching policy is widely discussed and it can be further divided into two parts, namely the cache permission policy and the cache replacement policy. The former is used to decide whether an incoming content should be cached while the latter is used to evict a cached content if required.

The Internet is a user-oriented network and popular contents always have much more requests than unpopular ones. Caching such popular contents closer to the user's location can improve the network performance, and consequently, the local caching policy is required to identify popular contents. However, considering the line speed requirement of ICN routers, the local caching policy whose complexity is larger than O(1) cannot be applied. In terms of the replacement policy, Least Recently Used (LRU) is selected as the default one for ICN because of its low complexity, although its ability to identify the popular content is poor. Hence, the identification of popular contents should be completed by the cache permission policy.

In this paper, a cache permission policy called Cache-Filter, whose complexity is O(1), is proposed, aiming to store popular contents closer to users. Cache-Filter takes the content popularity into account and achieves the goal through the collaboration of on-path nodes. Extensive simulations are conducted to evaluate the performance of Cache-Filter. Leave Copy Down (LCD), Move Copy Down (MCD), Betw, ProbCache, ProbCache$^+$, Prob(p) and Probabilistic Caching with Secondary List (PCSL) are also implemented for comparison. The results show that Cache-Filter performs well. For example, in terms of the distance to access to contents, compared with Leave Copy Everywhere (LCE) used by Named Data Networking (NDN) as the permission policy, Cache-Filter saves over 17% number of hops.

Keywords: Information Centric Networking; caching performance; the cache permission policy; the cache replacement policy; Least Recently Used.

This work is supported by National Basic Research Program of China (973 program) under Grant No. 2013CB329101, National High Technology of China (863 program) under Grant No. 2015AA015702, and National Natural Science Foundation of China under Grant No. 61271202, No. U1404611 and No. 61471029.

1. Introduction

As the number of Internet users and the scale of applications continue to grow, many serious drawbacks of the current Internet are exposed, such as the poor scalability and the low resource utilization. How to design the next-generation network has become one of the hottest issues in academic communities. Among the proposals, Information Centric Networking (ICN) has recently attracted great attention, with some well-known research projects and approaches including NDN [1], PURSUIT [2], COLOR [3] and so on [4-7]. ICN shifts from the original host-based network to a new content-centric one, and aims at creating a more efficient content distribution network. Specifically, ICN names the content decoupling from its location and introduces the network caching, thus, contents can be cached anywhere within the network, with accompanying advantages such as reducing the user delay and offloading the traffic from the server.

The benefits of introducing the network caching are obvious but there are still many challenges needed to be solved, including the local caching policy issue [8-17], the cache collaboration policy issue [18-20], the cache allocation policy issue [21-22], etc. In this paper, we mainly focus on the cache permission policy, which is part of the local caching policy. The local caching policy can be divided into two parts, namely the cache permission policy and the cache replacement policy. The cache permission policy is used to determine whether the new incoming content should be allowed to enter the cache while the cache replacement policy is used to determine which cached content should be removed to make room for the new one when the cache is full.

The Internet is a user-oriented network and popular contents always have much more user requests than unpopular ones. Caching such popular contents closer to users can improve the network performance. The traditional counter-based local caching policies that need to record the number of requests can effectively identify popular contents, however, as they perform the sorting operation, their complexity is at least O(logN). Considering the line speed requirement of ICN routers [23-24], the local caching policy whose complexity is larger than O(1) cannot be applied. As far as the cache replacement policy is concerned, Least Recently Used (LRU) is selected as the default cache replacement policy in ICN [25], since it has almost the best performance among policies whose complexity is O(1) such as First In First Out (FIFO) and randomized policies. Nevertheless, its ability to identify the popular content is poor. LRU can be regarded as a one-time request history replacement policy and cannot directly reflect the number of requests. Thus, the function of identifying popular contents according to their number of requests should be completed by the cache permission policy.

As the network caching is shared by all contents and the average size of a content is much larger than that of the web page, the replacement rate of cached contents in cache nodes will be extremely high and contents cannot be cached steadily. Many contents have been deleted before their subsequent requests arrive, making popular contents to be easily replaced by unpopular ones. Consequently, the caching effect of improving the network performance cannot be well reflected. Hence, an effective cache permission policy that can reflect the content popularity is required, and its complexity should be O(1). That is the basic design principle of our proposed Cache-Filter.

Similar as LCD, Cache-Filter moves a new copy of content down a hop when the request hits the cache. After reaching the downstream node, the name of the content is first recorded in CacheFilter (CF) list instead of the content itself. Only when the content flows through the node again and the name of the content is not removed by CF list will the content be cached. CF list actually is used for screening popular contents and reducing the frequency of replacing cached contents, thus, cached contents are guaranteed to be stable in cache nodes.

In order to evaluate the performance of our proposed Cache-Filter, simulations are conducted under a 6-layer tree and a 625-node Internet-like topology. For performance comparison, LCD [8], Move Copy Down (MCD) [8], Prob(p) [8], Betw [9], ProbCache [10], ProbCache$^+$ [11] and Probabilistic Caching with Secondary List (PCSL) [12] are also implemented. The results show that in terms of the distance to access to contents, compared with Leave Copy Everywhere (LCE) that is used by NDN as the permission policy, Cache-Filter saves over 17% number of hops.

The paper is organized as follows. In Section 2, related works of cache permission policies are given, while in Section 3, the problem description is presented to introduce the motivation of Cache-Filter. The detailed mechanism of Cache-Filter is illustrated in Section 4 and the simulation results are shown in Section 5. The last section is the conclusion.

2. Related Work

The cache issue has been intensively studied in the field of web caching [26], Content Delivery Network (CDN) [27] and IPTV [28], etc., and the local caching policy is always one of the hot topics widely discussed. Different from previous studies that are aimed at certain specific applications, ICN faces to all network contents, making the network caching become a sort of public competed resource.

The local caching policy can be divided into the cache permission policy and the cache replacement policy. The latter has gained lots of attention in web caching. Many schemes have been proposed, considering the request frequency, the content size, the content age, etc. The literature [29] provided a very detailed summary. However, due to the line speed requirement of ICN routers, the complexity of the cache replacement policy used in ICN is required to be O(1), making LRU as the default option. Consequently, previous works for the local caching policy in ICN are mainly focused on the cache permission policy.

The work in [8] proposed three cache permission policies including LCD, Move Copy Down (MCD) and Prob(p). LCE is also used to make comparison. Those policies were designed for web caching, but can also be applied to ICN. LCE can be regarded as an extreme permission policy that allows all contents to be cached, which is currently adopted by NDN. LCE leaves one copy of content on every en-route node in order to satisfy subsequent requests. Nevertheless, in web caching, the size of a web page is pretty small and the cache capacity is not the key factor to limit the cache performance. Different from the web caching, ICN network caching is a sort of public resource that are shared by all contents. In addition, ICN routers are required to forward contents at the line speed, making the cache capacity subject to the bandwidth of Network Interface Card (NIC). Consequently, the ratio of the cache capacity and the content number will be rather small. Moreover, ICN has interactions among caching nodes. Using LCE as the permission policy will cause the same content to be cached by all en-route nodes, wasting the limited cache resources and increasing the cache replacement rate significantly. Thus, LCE might not be the best choice for ICN.

LCD and MCD can roughly reflect the number of requests. Under LCD and MCD, a new copy of the requested content will be cached only at the immediate downstream node of the hit node. Thus, contents that have more requests will be closer to users. The difference between LCD and MCD is that under MCD, the requested content will be removed by the hit node, reducing the redundancy of cached contents along the delivery path. LCD and MCD make use of the number of requests to screen popular contents, however, the ratio between the request frequency and the path length will affect their performance.

Prob(p) is a probability-based cache permission policy. When the content is sent back, the en-route node will cache the content at a fixed probability *p*. Compared with LCE, Prob(p) increases the diversity of cached contents and decreases the content replacement rate of cache nodes along the delivery path.

Betw [9] is aimed to cache contents at important nodes where a cache hit is most likely to happen, so as to achieve the requirement of *cache less for more*. Their solution is based on the concept of Betweenness Centrality (BC) which measures the number of times a specific node lies on the content delivery paths between all pairs of nodes in a network topology. As the request is sent upstream, the maximum value of BC of the node along the delivery path will be recorded in the request. When the request reaches the server or the node that caches the content, the value of BC in the request will be attached to the content header. As the content is sent back, each on-path node will compare its own BC value with the one containing in the content header. Only when the two BC values are equal will the node cache the content. Although Betw stores the content on a node that is more likely to produce a cache hit, it will accelerate the content replacement rate of those nodes, resulting in the instability of cached contents. Meanwhile, BC is a topological property and does not capture the content popularity.

I. Psaras et al. proposed two probability-based cache permission policies called ProbCache [10] and ProbCache$^+$ [11]. ProbCache$^+$ is the enhanced version of ProbCache. The idea behind them is caching contents closer to users. The probability also takes the remaining cache capacity of the delivery path into account. However, the content popularity which is the most crucial factor for caching performance is not considered. The caching probability expressions of ProbCache and ProbCache$^+$ for node X to cache the incoming content are shown in Eq. (1) and Eq. (2) respectively, where c represents the total length of the delivery path, x is the distance between the node X and the hit node, N_i is the cache capacity of the i^{th} node on the delivery path from the user side, N_X is the cache capacity of the node X and T_{tw} equals 10.

$$\text{ProbCache:}\quad \mathrm{P}(X) = \frac{\sum_{i=1}^{c-(x-1)} N_i}{T_{tw} N_x} \cdot \frac{x}{c} \tag{1}$$

$$\text{ProbCache}^{+}\text{:}\quad \mathrm{P}(X) = \frac{\sum_{i=1}^{c-(x-1)} N_i}{T_{tw} N_x} \cdot \left(\frac{x}{c}\right)^{c} \tag{2}$$

Garcia-Reinoso et al. proposed two caching policies called LRU-PC and LRU-PCSL [12]. The former is actually the combination of Prob(p) and LRU. The latter is the update version of LRU-PC with a Secondary List (SL). When the content is sent back, the content name is firstly cached in SL with a fixed probability p_{sl}. If the content is already in SL, then, the content is cached in the cache with a fixed probability p_c. This idea is similar with ours, however, we take the content popularity into account and the length of CF list is much less than that of cached content list, which is just the opposite as LRU-PCSL does.

K-LRU [13] records the last K references of cached contents. Although it is a caching replacement policy, K-LRU also uses a new list to filter contents to be cached before they are actually cached. However, since K-LRU requires to keep track of the time of last K references of the content, its complexity is no longer as O(1) as that of LRU.

Tarnoi et al. compared the performance of different combinations of cache permission policies and cache replacement policies in the work [14] and found that the combination of Prob(0.01) and LRU has a good result.

An age-based permission policy called ABC was proposed in the work [15]. Under the policy, the more popular the content is and the closer the content is to users, the longer the cache age will be allocated to the content. Once the content gets the allocated age, it cannot be removed until the age is expired. Suksomboon et al. proposed a probability-based permission policy that takes the content popularity into consideration [16]. However, both of above works assume that the content popularity is known in advance by some other means, which requires some additional complex operations. Bernardini et al. proposed a popularity-based permission policy that records the content request number [17], but it does not meet the requirement of the line speed.

The comparison of above mentioned permission policies is shown in **Table 1**. Cache-Filter, LCD, MCD, Betw, ProbCache and ProbCache$^+$ add a tag in the packet and have to deal with it when the content is sent back. Prob(p), ProbCache, ProbCache$^+$, PCSL, and PopCache are supposed to produce a random number to compare with a defined or calculated probability. ABC calculates the cache time for the incoming content based on the content popularity and MPC counts the request number for the content. In addition, Cache-Filter and PCSL firstly record the content name instead of the content itself.

Table 1. The comparison of related caching permission policies

Permission Policy	*Factors to be considered*	*Compl-exity*	*Packet Tag*	*Operations on*
Cache-Filter	*Content Popularity, Content Name List*	*O(1)*	√	*Tag and Name List*
LCD[8]	*Content Popularity*	*O(1)*	√	*Tag*
MCD[8]	*Content Popularity*	*O(1)*	√	*Tag*
Prob(p)[8]	*Probability*	*O(1)*	×	*Probability Calculation*
LCE	---	*O(1)*	×	---
Betw[9]	*Topologic Features*	*O(1)*	√	*Tag*
ProbCache[10]	*Distance, Probability, Cache Capacity*	*O(1)*	√	*Tag and Probability Calculation*
ProbCache$^+$[11]	*Distance, Probability, Cache Capacity*	*O(1)*	√	*Tag and Probability Calculation*
PCSL[12]	*Probability, Content Name List*	*O(1)*	×	*Probability Calculation and Name List*
ABC[15]	*Content Popularity, Distance, Cache Time*	*O(1)*	×	*Cache Time Calculation (Content Popularity is known in advance)*
PopCache[16]	*Content Popularity, Probability*	*O(1)*	×	*Probability Calculation (Content Popularity is known in advance)*
MPC[17]	*Content Popularity*	*O(logN)*	×	*Counts of Request Number*

3. Problem Description

Unlike the previous studies of caching, ICN is the content-oriented network and all contents within the network can make use of the cache resources. Thus, how to allocate the cache resources for different contents so as to reduce the user delay and network traffic is the main goal of designing the local caching policy. Although this issue has been thoroughly studied in other fields such as the web caching and CDN, those solutions cannot be directly applied to

ICN. Since ICN regards the content as the first class network citizens, the traditional channel-based connection is replaced by the hop-by-hop transmission. Many new features have been generated. The most crucial feature for designing the cache permission policy should be the line speed requirement for ICN routers. It causes the following results.

1) The complexity of the cache permission policy should be O(1) and LRU has become the default cache replacement policy in ICN. Complex policies cannot be used, although their performance is better.

2) The cache capacity cannot be enlarged arbitrarily and it is related to the bandwidth of the NIC. In addition, all contents compete the caching resources in the network, making the ratio of the cache capacity and the content number very small. The work [25] even estimated that this ratio is at an order of magnitude of 10^{-5}.

The current Internet is a user-oriented network and the content popularity is the most crucial factor for the caching performance [25]. Popular contents have much more user requests than unpopular ones. However, LRU is a kind of replacement policy that only records one-time history. Thus, it can guarantee the timeliness of cached contents and avoids the cache pollution problem that Least Frequently Used (LFU) suffers. On the contrary, LRU cannot identify popular contents since it does not count the request number, making its performance worse than that of LFU.

As a cache replacement policy, LRU operates like below. For simplicity, we treat LRU as a list that consists of many entries. Each entry contains the content name and some related information like the address where the content is cached. When a new content enters the cache, a new entry will be added to record the content name and related information at the top of the list. If the cache is full, the entry at the bottom of the list will be removed. When the cached content is hit, the corresponding entry in the list will be moved to the top. Suppose that the LRU list contains at most n entries. A popular content c is requested at time t_1, and requested again at t_2. The condition which makes the content c to be remained in the LRU list is that other requests produced during the time interval (t_2- t_1) are subscribed for not more than n different contents. However, since the ratio of the cache capacity and the content number is pretty small, with high total request frequency or long interval (t_2- t_1), the content can be easily evicted by others. Thus, the stability of popular contents cannot be guaranteed, which certainly degrades the cache performance.

In addition, cache nodes in ICN are interacted. In the delivery path, the upstream nodes are directly affected by the downstream nodes because many requests are satisfied by the downstream nodes. The instability of the downstream cached contents will cause more drastic changes to upstream nodes. Besides, if LCE is used as the cache permission policy which makes all nodes along the delivery path cache the same content, the overall content replacement rate of cache nodes will be much higher. A replacement error, which means the popular content is evicted by the unpopular one, will cause all nodes on the path to cache the same unnecessary contents and the diversity of cached contents along the delivery path becomes very poor. These all indicate that LRU, as the default cache replacement policy, does not perform well in ICN.

Therefore, the content popularity or the number of requests should be somehow reflected by the cache permission policy. Moreover, the complexity of the cache permission policy should be O(1).

4. The Proposed Cache-Filter

4.1 The overview of Cache-Filter

Cache-Filter is a cache permission policy whose complexity is O(1) and it takes the content popularity into account. It aims to 1) cache popular contents closer to users so as to reduce the user delay and the network traffic; 2) decrease the content replacement rate of cache nodes in order to keep cached contents steadily; 3) ensure the diversity of cached contents along the delivery path to improve the cache efficiency.

As the complexity of policies that involve the sorting operation is at least O(logN), it is not feasible for the node to determine the content popularity by counting the number of requests for some contents. Thus, the number of requests has to be reflected through the mutual cooperation of the on-path nodes. Under Cache-Filter, each request may advance a new copy of the content one hop closer to the user. A flag called CacheFlag is added in the content header. Its default value is set as false. The false means that the node is not allowed to cache the content if its cache is full while the true means just the opposite. Only the server or the hit node that holds the requested content can change CacheFlag to be true. After the content is sent to the immediate downstream node, CacheFlag in the content header will be set as false so that other nodes in the delivery path cannot cache the content anymore if their caches are full.

Since the ratio of the cache capacity and the content number is extremely small, a new content to be cached will result in another content to be excluded. As mentioned above, LRU, as the cache replacement policy in ICN, cannot identify the content popularity well and leads to the high content replacement rate of cache nodes. In order to alleviate these problems, the node using Cache-Filter as the cache permission policy does not immediately cache the content whose CacheFlag is true. Instead, it just records the content name in a CacheFilter (CF) list and waits for a moment to see whether the content is popular. If the same content whose CacheFlag is true is sent back again and the content name is still in CF list, then, the node will cache the content locally.

CF list uses LRU as the list replacement policy so that its complexity can be limited at O(1). As mentioned above, LRU, as the cache replacement policy, has a high content replacement rate under ICN environment and cannot identify the content popularity well. However, CF list just utilizes this feature of LRU to filter out unpopular contents from the server side. In fact, CF list transfers the replacement rate from the cache replacement policy to the cache permission policy to ensure the stability of cached contents. Moreover, the length of CF list is set much less than that of the cached content list like ContentStore (CS) in NDN, not only accelerating the replacement rate in CF list but also reducing the node overhead. Therefore, under a high CF list replacement rate, the content which is requested again before its name is evicted by CF list is very likely to be the popular one. Then, after the same operation of other on-path nodes, only the popular contents will be cached close to users.

Cache-Filter screens popular contents from the source side, and the more popular the content is, the quicker it is pulled down to the network edge and the closer it is to the users. Thus, the replacement error caused by LRU can be well alleviated since the content popularity of cached contents along the delivery path is in a descending order from the user side, guaranteeing the stability of cached contents along the delivery path.

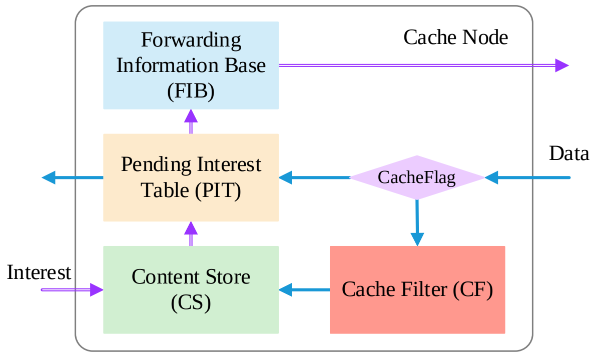

Fig. 1. The flow chart of Cache-Filter

The general flow chart of Cache-Filter is illustrated in **Fig. 1** and NDN is selected as the ICN frame. In the following, we use the terms request and *Interest*, content and *Data* interchangeably. The flow of processing *Interest* is almost the same as NDN. When *Data* is sent back, if its header contains a true CacheFlag, the content name will be firstly recorded in CF list. Only after being hit in CF list, the content can be cached. CF list takes effect when *Data* is sent back and can avoid conflict with CS. As *Interest* is sent upstream, the node only looks up its CS and CF list is not involved. As *Data* is sent back, the node will firstly look up its CF list.

4.2 The operation of Cache-Filter

This section describes the operation of Cache-Filter in detail and the related pseudo-code is shown in **Table 2**.

As *Interest* is sent upstream, if it reaches the server or the node that caches the corresponding content, *Data* will be returned and CacheFlag in *Data* header is set to true. Otherwise, the node checks PIT. If the content name does not exist, the node will create a related entry and forward *Interest* according to FIB. Otherwise, the node will suppress *Interest* to be sent upstream.

When *Data* is sent back, the node will firstly check CacheFlag. If the flag is false, *Data* is unnecessary to be cached. However, during the network initialization, it takes some time for Cache-Filter to make full use of the cache capacity. Thus, the node can randomly cache *Data* if its cache is not full. In this case, the node produces a random number A (rnd_A). If rnd_A is larger than 0.5, *Data* will be cached locally. Since this *Data* has not been screened by CF list, it will be placed at the bottom of CS and will be the first to be removed if required, in case that it is not a popular content.

If CacheFlag of *Data* is true, *Data* will be likely to be cached. The node firstly changes CacheFlag to false and then, checks CF list. If the name of *Data* exists in CF list, the node caches *Data* in CS and removes the corresponding entry in CF list. Meanwhile, if the cache is full, there will be another content to be replaced. The replaced content name will be stored in CF list in case that it is a popular content. In order to increase the diversity of cached contents along the delivery path, a Remove Message whose Time-to-Live (TTL) is one will be sent by the node to its immediate upstream node where the content comes from, notifying that *Data* has been cached downstream. On receiving this Remove Message, the upstream node will check its degree (the number of neighbors). If the degree value is not more than 2, the corresponding content will be removed. Otherwise, the node just drops the Remove Message. The motivation is that it is unnecessary to cache the same content along an exclusive path.

If the name of *Data* is not included in CF list, a corresponding entry will be created in CF list.

In this case, if the cache is not full, the node produces a random number *A* (*rnd_A*). If *rnd_A* is larger than 0.5, *Data* is cached locally and the corresponding CF entry is removed. Finally, *Data* is distributed according to the related PIT entry.

Note that the complexity of our proposed Cache-Filter is O(1). As stated before, CF list takes effect only when contents are arrived from the upstream and it screens popular contents before CS caches contents. In fact, CF list and CS work in a strictly sequence, not conflicting with each other. Hence, there is no extra complex operation required between them. The complexity of Cache-Filter mostly depends on CF list, more accurately, LRU, the replacement policy of CF list. As for the implementation of LRU whose complexity is O(1), the core idea is to maintain two separate data structures, a hash table which to check whether the requested content is in LRU list, and another is a doubly linked list which to keep the sequence of the cached contents. When *Interest* is coming, if the requested content is cached, by checking the hash table, LRU can find the location of the requested content in the doubly linked list and send the content back. Then, LRU links the previous content of the requested content to the next content of it. Finally, LRU puts the requested content at the head of the doubly linked list. When *Data* is coming, LRU adds the content name to the hash table and the head of the doubly linked list. Besides, if the cache is full, the content name at the tail of the doubly linked list should be removed, together with the corresponding entry in the hash table.

Table 2. The pseudo-code of the operation of Cache-Filter

The Operation of CacheFilter

```
1:  Funciton:ReceiveInterest (Interest(c), inface)
2:  c=Interest(c)->GetContentName( );
3:  CS_Hit=CS->Lookup(c);
4:  If (CS_Hit)
5:  | Data(c)=GetData(c);
6:  | Data(c)->SetCacheFlag (True);
7:  | Forward (Data(c), inface);
8:  Else
9:  |  Exist=PIT->Lookup(c);
10: |  If (!Exist)
11: |  | PIT->Create (c, inface);
12: |  | Forward (Interest(c), FIB);
13: |  Else
14: |  | Suppress (Interest(c));
15: |  End If
16: End If
17:
18: Function: ReceiveData (Data(c), inface)
19: c=Data(c)->GetContentName( );
20: Flag=Data(c)->GetCacheFlag( );
21: If (Flag==False)
22: | If (!CacheFull && (rnd_A>0.5))
23: | | CS->Add(Data(c), back);
24: | End if
25: Else
26: | Data(c)->SetCacheFlag (False);
27: | CF_Hit=CF->Lookup(c);
28: | If (CF_Hit)
29: | | CS->Add(Data(c), front);
30: | | CF->Remove(c);
31: | | If (CacheFull)
32: | | | CF->Add(c', front);
33: | | | #c': the name of the evicted content
34: | | Endif
35: | | SendRemoveMessage(c, inface);
36: | Else
37: | | CF->Add(c, front);
38: | | If (!CacheFull && (rnd_A>0.5))
39: | | | CS->Add(Data(c), back);
40: | | | CF->Remove(c);
41: | | End if
42: | End if
43: End if
44: Forward (Data(c), PIT);
45:
46: Function: ReceiveRemoveMessage (c, inface)
47: If (node->GetDegree( )<3)
48: | CS->Remove(GetData(c));
49: End if
```

4.3 A simple example

A simple example of how Cache-Filter operates is provided below. The running statuses of the node A and B are illustrated in **Figs. 2(a) ~ 2(f)**. Assume CS and CF list have 3 and 2 entries respectively and the request sequence is {#1, #2, #1, #3, #1, #1}.

The user request for the content #1 is depicted in **Fig. 2(a)**. The server sets the CacheFlag of the content #1 to true (T) and sends it back. After receiving the content, the node B changes the CacheFlag to false (F) and records the content name in CF list. Suppose the produced rnd_A is less than 0.5 in both the node A and B, then, the content #1 is not cached along the delivery path.

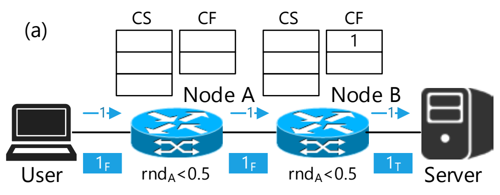

Fig. 2(a). Requesting for content #1

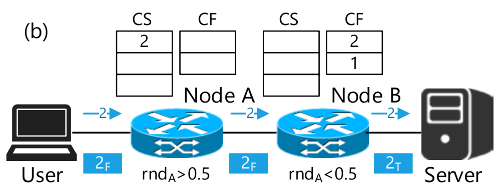

Fig. 2(b). Requesting for content #2

The user request for the content #2 is depicted in **Fig. 2(b)**. Suppose the produced rnd_A is larger than 0.5 in node A, then, the content #2 is cached by the node A.

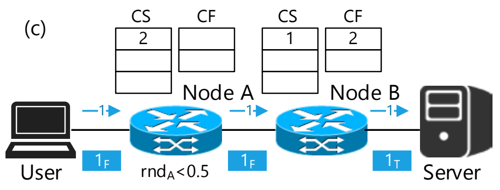

Fig. 2(c). Requesting for content #1

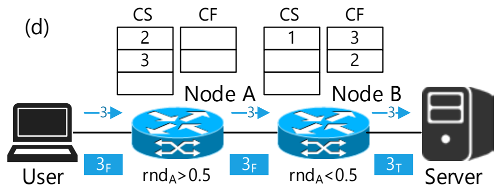

Fig. 2(d). Requesting for content #3

The user request for the content #1 is depicted in **Fig. 2(c)**. Since content #1 is stored in CF list of the node B, it will be cached and the related CF list entry is removed.

The user request for the content #3 is depicted in **Fig. 2(d)**. The content #3 is inserted in CF list of the node B. Suppose the produced rnd_A is larger than 0.5 in node A, then, content #3 is cached by the node A and it is inserted at the bottom of CS because the node A caches content #3 in a random way.

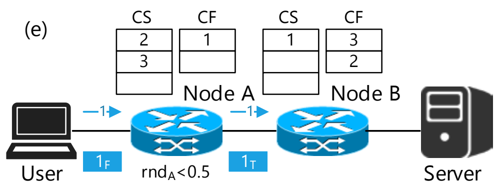

Fig. 2(e). Requesting for content #1

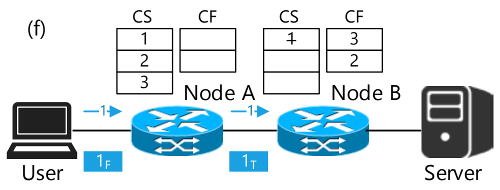

Fig. 2(f). Requesting for content #1

The user request for the content #1 is depicted in **Fig. 2(e)**. Since content #1 has been cached by the node B, it is sent back with CacheFlag true by the node B. Then, the content #1 is inserted in CF list of the node A.

The user request for the content #1 is depicted in **Fig. 2(f)**. Since content #1 exists in CF list of the node A, after receiving the content, the node A will cache content #1 in CS and remove related entry in CF list. Then, it will send a remove message to the node B and notify the node B that it has cached the content #1. On receiving that message, the node B removes the content #1 to make room for other contents.

4. Simulation Results

The function of Cache-Filter (C-F) is implemented with ndnSIM [30], a NS-3 based NDN simulator, and additionally, some cache common permission polices whose complexity is O(1) are also implemented for performance comparisons. Those policies include LCD, MCD, Betw (BTW), ProbCache (PRC), ProbCache$^+$ (PRC$^+$), Prob(0.01), Prob(0.3), Prob(0.7), and PCSL. LRU is selected as the cache replacement policy for them. The combination of LCE and LRU is the local caching policy of NDN, which has been provided by ndnSIM. Note that according to the work [12], p_c and p_{sl} in PCSL are set at their best values, 0.1 and 0.5 respectively, and the length ratio of SL and CS is 10.

All permission policies are run under a 6-layer tree and 625-node Internet-like topology to investigate their performance. The total content number is set to 10,000, and each content has the same size. The content requests follow the Zipf distribution where the skewness parameter ranges from 0.5 to 1.5, while the arrival process of content requests follows the Poisson distribution where the arrival rate is 10Hz for each user. The cache capacity is set to contain 50 contents by default and it ranges from 10~100 contents. The length of CF list is set to 20% of the cache capacity by default. The initial cache is empty. The simulation duration lasts 11,000 seconds where the first 1,000 seconds is for the warm-up. The simulation related parameters are summarized in **Table 3** and *U* in the row of *Local Policy* denotes the union symbol. Without otherwise specified, the cache capacity and the size of CF take the default values.

Table 3. The list of related parameters

Parameter	*Value*
α of Zipf distribution	*0.5~1.5, 1.0 by default*
Topology	*Tree Topology (64 nodes), Internet-like Topology (625 nodes)*
Frequency of each user	*10Hz*
Content Number	*10,000*
Simulation Duration	*11,000 seconds*
Cache Capacity	*10~100, 50 by default*
The length ratio of CF list and CS	*10%~100%, 1,000%, 20% by default*
Local Policy	*{C-F, LCD, MCD, BTW, PRC, PRC+, Prob(0.01), Prob(0.3), Prob(0.7), PCSL, LCE} U {LRU}*
Performance Metric	*Distance (hops), Server Hit Ratio, Cache Eviction Counts*

As for the topology, the 6-layer tree topology consists of 64 nodes. 32 users are connected to different leaves and a server is accessed to the root, thus, the distance between each user to the server is 7 hops. The bandwidth of each node in the tree topology is set at 10Gbps.

The Internet-like topology is selected from the Rocketfuel's AT&T topology [31], as shown in **Fig. 3**. It contains 625 nodes and 2,101 links. Specifically, there are 296 leaves (red), 108 gateways (green) and 221 backbones (blue). We do not change the original parameters of the Internet-like topology. In the Internet-like topology, there are 100 users and 100 servers randomly accessing at leaves. 10,000 contents are arbitrarily allocated to those 100 servers but each content is owned by a unique server. We change the random seed and repeat the simulation 20 times under the Internet-like topology, resulting in 20 kinds of distribution scenarios. Since the change trend of each cache permission policy is very similar in all scenarios, we only present results of the first scenario.

The testing performance indexes include *Distance (Dist)*, *Server Hit Ratio (SHR)* and *Cache Eviction Counts (CEC)*. The *Distance* is defined as the number of hops that the request reaches the server or the node which has the requested content copy. The *Average Distance* is the quotient of the summation of the *Distance* of each request and the total number of requests. The shorter the average distance, the higher the efficiency of the network caching performance is. Note that we do not use the user delay as a performance metric because it is hard to judge the cause of a long delay if the link delay itself is quite long in the topology. *Average Server Hit Ratio* is defined as the ratio of the total request number received by all servers and the total requests number sent by all users. It roughly shows the ability that the network caching as a whole can unload the server traffic. Finally, *Cache Eviction Counts* is defined as the number of times that the cached content is evicted from all nodes in the network, which reflects the stability of the cached contents. In addition, the term *Transit* is regarded as per second. Without otherwise specified, parameters are set at their default values.

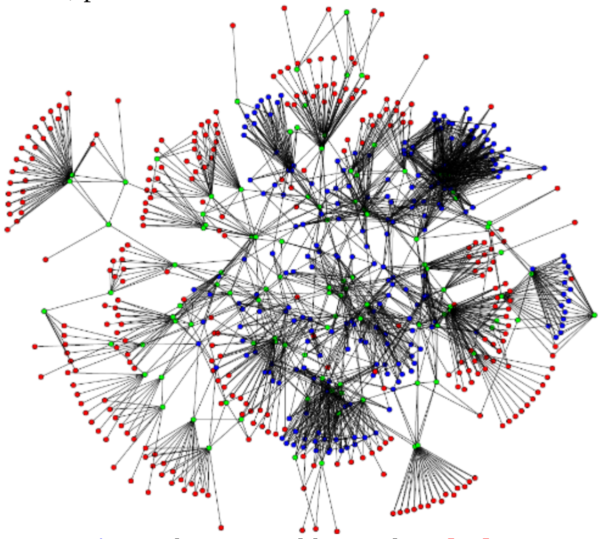

Fig. 3. The Internet-like topology [31]

5.1 The length of CF list

First, the length of CF list is investigated to see how it influences the performance of Cache-Filter. The cache capacity is set to 50 and the length of CF list ranges from 5 to 50 entries. CF list containing 500 entries is also evaluated. The relationship between *Average Distance* within the last 10,000 seconds and the length of CF list is illustrated in **Fig. 4**.

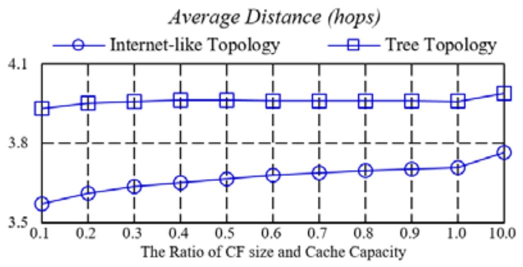

Fig. 4. The relationship between Average Distance and the length of CF list

It is observed that *Average Distance* increases slightly as the length of CF list enlarges under both of the topologies. Since LRU cannot identify popular contents well, larger length of CF list will reduce the replacement rate of CF list and cause more content names to be recorded in CF list. Thus, contents that are not very popular have more opportunities to be inserted in CF list and then to be cached by nodes, resulting in the improvement of the content replacement rate and degrading the cache performance lightly. In addition, larger length of CF list requires more cache resources overhead, hence, the length of CF list is set to 20% of the cache capacity by default. Note that the gap between two curves is the result of the different setting and characteristics of the two topologies.

5.2 Average Distance

Average Distance reflects the efficiency of the network caching and good cache permission polices can cache popular contents closer to users. On one hand, the user delay is reduced, improving the user experience. On the other hand, most of requests are satisfied by the network edge, dramatically decreasing the network traffic.

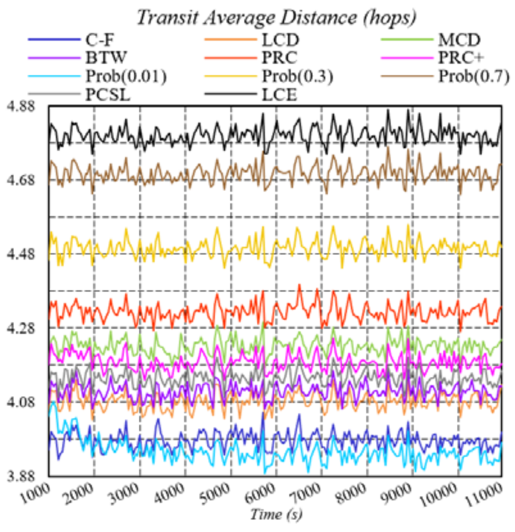

Fig. 5(a). Under the tree topology

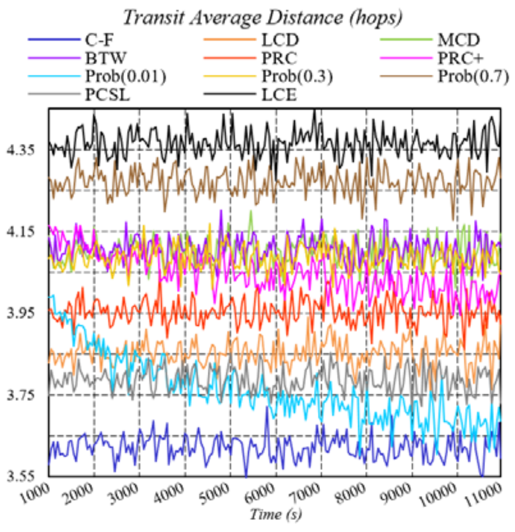

Fig. 5(b). Under the Internet-like topology

Transit Average Distance of each cache permission policy under the tree and Internet-like topology is illustrated in **Figs. 5(a) ~ 5(b)** respectively. Cache-Filter outperforms others except Prob(0.01) in the tree topology with the average of 3.98 and 3.62 number of hops respectively, while LCE performs the worst with the average of 4.80 and 4.37 number of hops. Compared with LCE, Cache-Filter saves over 17% number of hops. The detailed results of each permission policy about the average value within the last 10,000 seconds (AVG) and the reduction ratio (RTO) compared with LCE are listed in Tab. 4 and Tab. 5. Note that RTO compared with LCE is calculated as $(X_{Other\_Policy}-X_{LCE})/X_{LCE}$.

As stated before, since the ratio of the cache capacity and the content number is rather small in ICN and LRU is chosen as the cache replacement policy, it is not suitable to adopt LCE as the cache permission policy, which makes the content copy to be stored along the whole delivery path. LRU cannot identify popular contents as well as LFU, resulting in the instability of cached contents. Besides, LCE does not filter contents entering the cache, making the inherent drawback of LRU to be fully exposed. Thus, it makes sense that the combination of

LCE and LRU, which is used in NDN, performs the worst.

The main reason why all the other permission policies perform better than LCE is that, to some extent, they all filter out some contents into caches and decrease the content replacement rate of cache nodes. The performance difference among them is determined by the factors under consideration when setting the permission rule.

PCSL has good performance because it is the combination of Prob(p) and SL. Prob(p) can decrease the content replacement rate of cache nodes and SL can help to filter out popular contents. However, the setting of p_c and p_{sl} of PCSL will influence the performance of PCSL a lot and it requires much longer SL to keep track of content names.

Prob(0.01), Prob(0.3) and Prob(0.7) are a sort of cache permission policies purely based on probability. Prob(0.01) performs much better than Prob(0.3) and Prob(0.7), mainly because the former has much lower caching probability and can indirectly filter out popular contents. Prob(0.01) is no doubt the simplest to be implemented among other caching permission policies. Although it does not consider any other factors, Prob(0.01) is quite efficient in such an environment where the ratio of the cache capacity of each node and content number is really small, just as the work [14] illustrates. This is because by a very low probability, only popular contents that have many requests can be cached. However, besides the cache capacity of each node, the performance of Prob(p) with low probability also depends on the number of nodes in the topology, the skewness parameter of Zipf distribution, etc. In addition, it is observed that different from its good performance in the tree topology, *Transit Average Distance* of Prob(0.01) in the Internet-like topology even does not reach a steady state before 8,000 seconds. Since the tree topology used is a regular topology which all leaves are at the same depth, contents requested in nodes which locate at the same depth are almost the same. Thus, there is not much difference between the tree topology and a cascade topology which has the same length as the depth of tree topology. This is the reason why Prob(0.01) can quickly reach a steady state in the tree topology. In contrast, the Internet-like topology has 625 nodes, much larger than the depth of the tree topology and it takes a very long time for Prob(0.01) to reach its best performance. Considering contents in the real network is changed all the times, such a long time surly degrades the performance of Prob(0.01).

ProbCache and ProbCache$^+$ primarily take the length of delivery path into account while Betw considers the importance of the node. They all perform better than LCE, Prob(0.3) and Prob(0.7) since they are designed based on some features of the delivery path or the network characteristics. However, the content popularity, which dominates the cache performance, is ignored. LCD performs better under both topologies because it can reflect the content request number, with each request advancing a new copy of the content one hop closer to the user. However, LCD only makes use of the length of the delivery path to filter out unpopular contents. If the request frequency is high or the path length is short, its performance will be limited. MCD performs far worse than LCD because it removes the hit content after the content is forwarded. Contents that can be pulled down are very likely to be popular and deleting such contents will certainly degrade the content sharing.

The reason why Cache-Filter has better performance is described as follows. Considering the feature of ICN and the defect of LRU, we take the content popularity into consideration to design Cache-Filter. Besides, the complexity of Cache-Filter is ensured to be O(1). Cache-Filter just takes the feature of the high replacement rate of LRU to filter out unpopular contents. In order to prevent content from entering the cache immediately, it adds content name into CF list which uses LRU as the replacement policy. With a high replacement rate of CF list, the content whose name can be hit in CF list is probably a popular one. In addition, by the filter effect of all nodes along the delivery path from the server side, only popular contents can be pulled down to the edge of the network, which is also the reason why Cache-Filter is better than LCD. The outstanding performance of Cache-Filter under both of topologies indicates that the principle of caching popular contents closer to users works effectively.

The changes of *Average Distance* within the last 10,000 seconds of each cache permission policy as the cache capacity increases under the tree and Internet-like topology are illustrated in **Figs. 6(a) ~ 6(b)** respectively. It is observed that the change trend of each cache permission policy is generally the same except Prob(0.01) in the Internet-like topology. When the cache capacity enlarges, the performance of Prob(0.01) improves a little. The reason is that Prob(0.01) cannot identify popular contents. However, Cache-Filter still has better performance because it regards the content popularity as the key factor to design the policy. It is natural that *Average Distance* drops as the cache capacity increases, however, the rate of decline of each curve becomes slightly smaller. Since content requests follow the Zipf distribution, the lower-ranking content has substantially less request number than the higher-ranking one. Thus, it can be expected that when the cache capacity increases to a certain extent, continuing to enlarge the cache capacity of nodes does not improve the network performance, and this extent is associated with the skewness parameter of the Zipf distribution.

The changes of *Average Distance* within the last 10,000 seconds of each cache permission policy as α increases under the tree and Internet-like topology are illustrated in **Figs. 7(a) ~ 7(b)** respectively. It is observed that the change trend of each cache permission policy is generally the same. Cache-Filter still has better performance under the two topologies. *Average Distance* drops a lot as α increases since more requests are sent for less contents. It is also observed that the performance of Prob(0.01) in the tree topology is worse than that of Cache-Filter. This is because when α is relatively low, the difference of request number of the popular content is not as large as it is when α is relatively high and Prob(0.01) is hard to identify popular contents only by an extremely low probability. In addition, when α=1.5, even *Average Distance* of LCE is no more than 1.71, making the caching permission policy much less important than it is when α is relatively low.

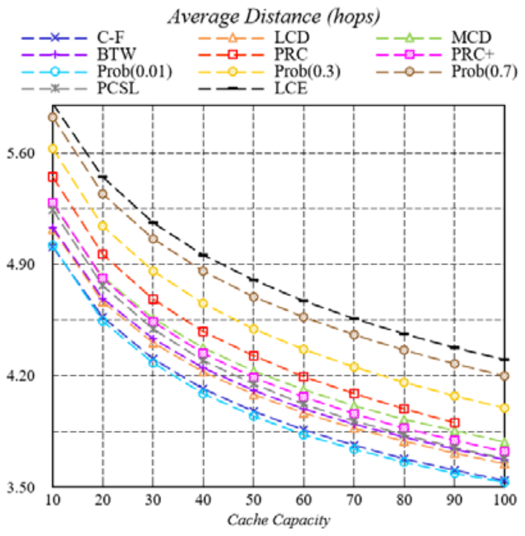

Fig. 6(a). Under the tree topology

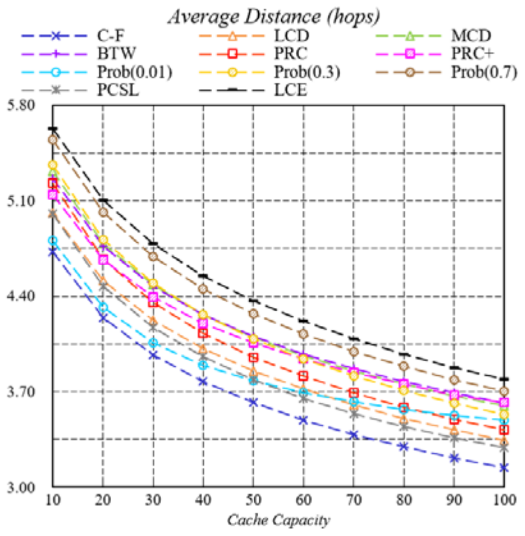

Fig. 6(b). Under the Internet-like topology

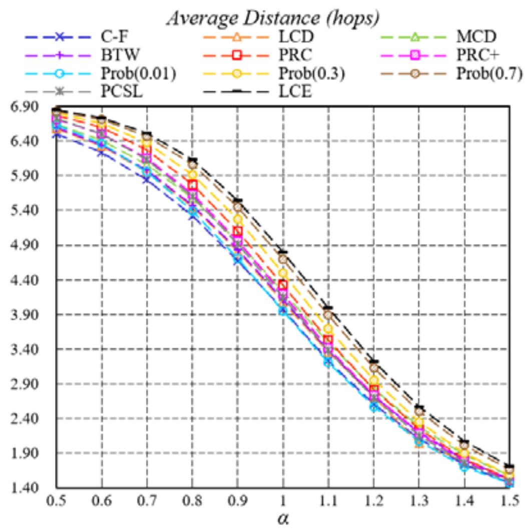

Fig. 7(a). Under the tree topology

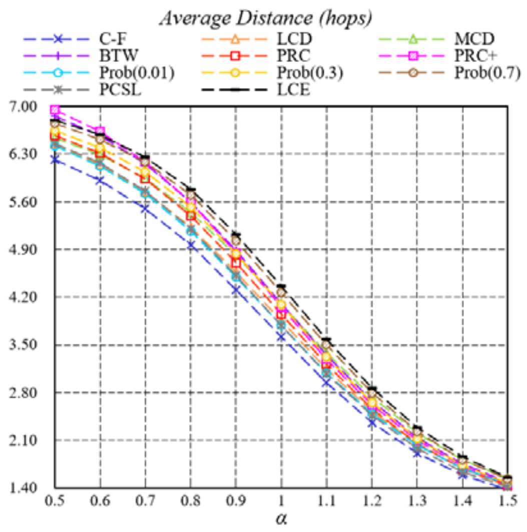

Fig. 7(b). Under the Internet-like topology

5.3 Server Hit Ratio

Transit Average Server Hit Ratio of each cache permission policy under the tree and Internet-like topology is illustrated in **Figs. 8(a) ~ 8(b)** respectively.

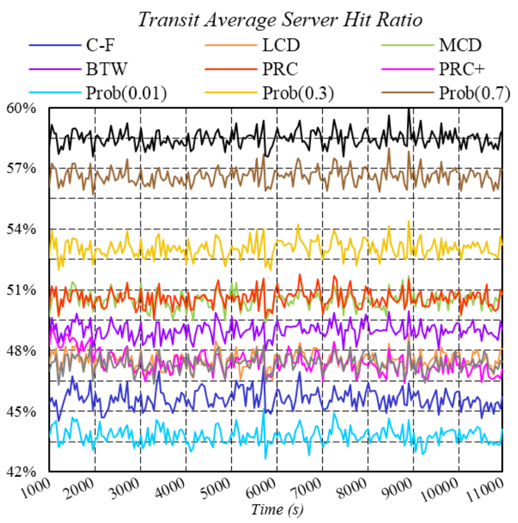

Fig. 8(a). Under the tree topology

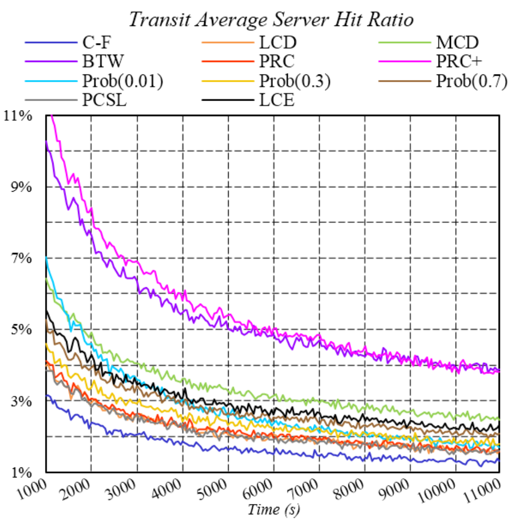

Fig. 8(b). Under the Internet-like topology

Cache-Filter has lower *Transit Average Server Hit Ratio* with 45.61% and 1.71% under the tree and Internet-like topology respectively while LCE performs poor with 58.47% and 2.97%. It is observed that the rank of *Transit Average Server Hit Ratio* of different permission policies is not consistent with the rank of *Transit Average Distance* of them. This is because the server hit ratio only reflects the caching performance of the network as a whole, however, how the cached contents are allocated along the delivery path cannot be reflected. For example, assume a very simple topology like *server-N1-N2-user* and each node can cache a content *c1* or *c2*. The server hit ratio remains the same in the situation where *N1* caches *c1*, *N2* caches *c2* and where *N2* caches *c1*, *N1* caches *c2*. Thus, *Average Distance* of each permission policy is necessary to be tested. It is observed that curves of *Transit Average Server Hit Ratio* of the Internet-like topology are quite different from that of the tree topology. The reasons are that the node number of the tree topology is much less than that of the Internet-like topology and that Internet-like topology is a scale-free one with 100 servers, making the majority of delivery paths to be crossed and overlapped. Therefore, it takes longer time for the Internet-like topology to reach a steady state.

As for why *Transit Average Server Hit Ratio* of Betw and ProbCache$^+$ is much higher than others, reasons are below. For Betw, since it caches contents at the node whose BC is the largest along the delivery path, contents in that nodes are easily evicted especially in the complex topology, making some less popular contents to be fetched from servers. For ProbCache$^+$, its cache probability almost entirely depends on the distance from users to hit nodes or servers, regardless of the content popularity, making popular contents have very low cache probability at following nodes of hit nodes. Thus, ProbCache$^+$ cannot guarantee that the more popular the content is, the more cache probability it can get, and popular contents are easily evicted. The detailed results of each permission policy about the average value within the last 10,000 seconds are listed in **Table 4** and **Table 5**.

5.4 Cache Eviction Counts

Transit Cache Eviction of each cache permission policy under the tree and Internet-like topology is illustrated in **Figs. 9(a) ~ 9(b)** respectively.

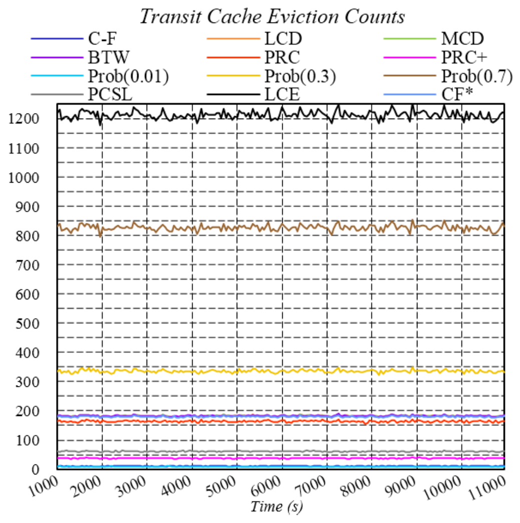

Fig. 9(a). Under the tree topology

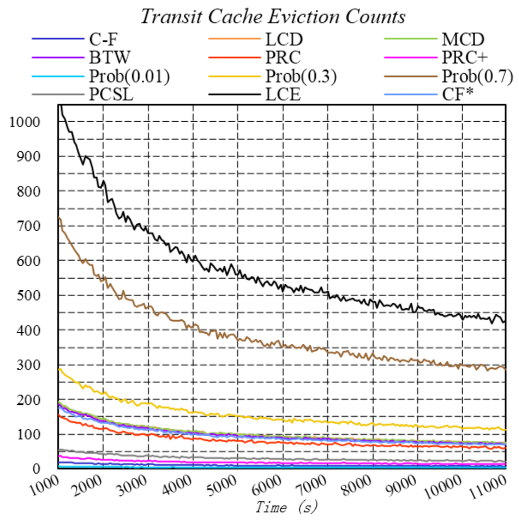

Fig. 9(b). Under the Internet-like topology

It is observed that *Transit Cache Eviction Counts* of LCE is extremely high, and *Average Cache Eviction Counts* of LCE within the last 10,000 seconds reaches 1213.11 and 578.11 per second. Thus, the stability of cached contents is not guaranteed and many contents are evicted before their subsequent requests are arrived. This has verified the argument we mentioned above that a very important reason why all the other permission policies perform better than LCE is that they all filter out some contents into caches and decrease the content replacement rate. Prob(0.01), Prob(0.3) and Prob(0.7) limit contents into caches by a fixed probability and the results of both *Average Distance* and *Average Cache Eviction Counts* are Prob(0.01)<Prob(0.3)< Prob(0.7)<LCE, indicating that it is very necessary to prevent some contents into caches to keep the cached contents steadily in ICN. Prob(0.01) has the best performance among others for its extremely low probability so that only popular contents with many requests can be cached.

Cache-Filter aims at reducing the content replacement rate and the reduction comes from two parts. One is produced by the effect of CacheFlag similar as LCD, which is a real reduction. Another is to transfer the replacement rate from the cache replacement policy to the cache permission policy. The eviction rate of CF list of Cache-Filter (CF*) is also shown in **Figs. 9(a) ~ 9(b)**. The average rate is at 179.90 and 95.06 in two topologies, although the length of CF list is set 20% of CS length to improve the replacement rate. It seems that the eviction rate of CF list should be much higher, however, due to the filter effect of first several nodes from the server side, only popular contents can be reserved at the network edge and the rest of nodes on the path seldom evicts contents in both CS and CF. Compared with LCE, Cache-Filter reduces the eviction rate at 99.13% and 98.20% under two topologies respectively. The detailed results of each permission policy about the average value within the last 10,000 seconds and the reduction ratio compared with LCE are listed in **Table 4 and Table 5**.

In summary, according to the simulation results, Cache-Filter has good performance because it takes the content popularity into consideration and leaves popular contents close to users. Cache-Filter uses the CF list and CacheFlag to filter out unpopular contents from the server side. By the filter effect of all nodes along the delivery path, only popular contents can be pulled down near users, so as to enhance the user experience, reduce network duplicate traffic and improve cache performance. Besides, Prob(p) with very low probability *p* is also a good caching permission policy for its simplification of implementation and low complexity. We will continue to optimize our Cache-Filter combined with Prob(p) in our future work.

Table 4. The detail results of each permission policy in the tree topology

Permission	*Dist*		*SHR*	*CEC*	
Policy	*AVG*	*RTO*	*AVG*	*AVG*	*RTO*
C-F	*3.98*	*17.15%*	*45.61%*	*10.61*	*99.13%*
LCD	*4.08*	*14.95%*	*47.53%*	*182.80*	*84.93%*
MCD	*4.23*	*11.84%*	*50.48%*	*182.71*	*84.94%*
BTW	*4.11*	*14.39%*	*48.98%*	*182.71*	*84.94%*
PRC	*4.32*	*9.93%*	*50.58%*	*163.92*	*86.49%*
PRC+	*4.19*	*12.75%*	*47.41%*	*37.67*	*96.89%*
Prob(0.01)	*3.95*	*17.75%*	*43.79%*	*9.55*	*99.21%*
Prob(0.3)	*4.50*	*6.36%*	*53.04%*	*335.67*	*72.33%*
Prob(0.7)	*4.70*	*2.15%*	*56.61%*	*826.89*	*31.84%*
PCSL	*4.15*	*13.59%*	*47.37%*	*61.26*	*94.95%*
LCE	*4.80*	*0*	*58.47%*	*1213.11*	*0*
*CF**		---		*179.90*	---

Table 5. The detail results of each permission policy in the Internet-like topology

Permission	*Dist*		*SHR*	*CEC*	
Policy	*AVG*	*RTO*	*AVG*	*AVG*	*RTO*
C-F	*3.62*	*17.07%*	*1.71%*	*10.39*	*98.20%*
LCD	*3.85*	*11.78%*	*2.15%*	*101.75*	*82.40%*
MCD	*4.10*	*6.10%*	*3.40%*	*102.31*	*82.30%*
BTW	*4.11*	*5.93%*	*5.29%*	*98.06*	*83.04%*
PRC	*3.95*	*9.44%*	*2.21%*	*82.64*	*85.71%*
PRC+	*4.06*	*7.03%*	*5.56%*	*19.01*	*96.71%*
Prob(0.01)	*3.78*	*13.41%*	*2.81%*	*4.26*	*99.26%*
Prob(0.3)	*4.09*	*6.42%*	*2.47%*	*156.02*	*73.01%*
Prob(0.7)	*4.27*	*2.15%*	*2.78%*	*390.40*	*32.47%*
PCSL	*3.79*	*13.24%*	*2.12%*	*31.21*	*94.60%*
LCE	*4.37*	*0*	*2.97%*	*578.11*	*0*
*CF**		*---*		*95.06*	*---*

5. Conclusion

In this paper, we have first analyzed some new generated features of ICN related to design a cache permission policy. Since LRU is selected as the default cache replacement policy and it cannot filter out unpopular contents well, the function of identifying content popularity is required to be completed by the cache permission policy and the complexity of the cache permission policy is supposed to be O(1). Then, we describe in detail the working mechanism of our proposed Cache-Filter. Specially, a flag called CacheFlag is attached in the content header and a CF list that uses LRU as the replacement policy is inserted into the nodes. In Cache-Filter, the incoming content is cached if its name is recorded in CF list and its CacheFlag is true. Through the cooperation of all on-path nodes, only popular contents can be pulled down to the network edge, reducing the user delay and the network traffic. Finally, extensive simulations are conducted under a 6-layer tree and a 625-node Internet-like topology to evaluate the performance of our proposed Cache-Filter. For comparison, LCD, MCD, Betw, Prob, ProbCache, ProbCache$^+$, and PCSL are also implemented. In terms of the distance to access to contents, compared with LCE that is used by NDN as the permission policy, Cache-Filter saves over 17% number of hops.

References

[1] V. Jacobson, D. K. Smelters, J. D. Thornton, M. F. Plass, N. H. Briggs and R. L. Braynard, "Networking Named Content," in *Proc. of the 5th international conference on Emerging networking experiments and technologies*, pp. 1-12, Dec. 2009. Article (CrossRef Link).

[2] N. Fotiou, D. Trossen and G. C. Polyzos, "Illustrating a publish-subscribe Internet architecture," *Telecommunication Systems*, vol. 51, no. 4, pp. 233-245, 2012. Article (CrossRef Link).

[3] H. Luo, Z. Chen, J. Cui and C. Qiao, "CoLoR: an information-centric internet architecture for innovations," *IEEE Network*, vol. 28, no. 3, pp. 4-10, 2014. Article (CrossRef Link).

[4] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren and H. Karl, "Network of Information (NetInf) – An information-centric networking architecture," *Computer Communications*, vol. 36, no. 7, pp. 721-735, 2013. Article (CrossRef Link).

[5] G. García, A. B'ben, F. J. Ramón, A. Maeso, I. Psaras, G. Pavlou, N. Wang, J. Sliwinski, S. Spirou, S. Soursos and E. Hadjioannou, "COMET: Content mediator architecture for content-aware networks," *Future Network & Mobile Summit (FutureNetw)*, pp. 1-8, Jun. 2011. Article (CrossRef Link).

[6] T. Koponen, M. Chawla, B. G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proc. of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 181-192, Aug. 2007. Article (CrossRef Link).

[7] Z. Yan, H. P. Chiang, Y. J. Park, X. Lee and Y. M. Huang, "Scalable and Secure Information-Centric Networking," *Journal of Internet Technology*, vol. 14, no. 6, pp. 867-880, 2013. Article (CrossRef Link).

[8] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta Algorithms for Hierarchical Web Caches," *IEEE International Conference on Performance, Computing, and Communications*, pp. 445-452, Apr. 2004. Article (CrossRef Link).

[9] W. K. Chai, D. He, I. Psaras and G. Pavlou, "Cache "less for more" in information-centric networks (extended version)," *Computer Communications*, vol. 36, no. 7, pp. 758-770, 2013. Article (CrossRef Link).

[10] I. Psaras, W. K. Chai and G. Pavlou, "Probabilistic In-Network Caching for Information-Centric," in *Proc. of the second edition of the ICN workshop on Information-centric networking*, pp. 55-60, Aug. 2012. Article (CrossRef Link).

[11] I. Psaras, W. K. Chai and G. Pavlou, "In-Network Cache Management and Resource Allocation for Information-Centric Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 2920-2931, 2014. Article (CrossRef Link).

[12] J. Garcia-Reinoso, I. Vidal, D. Diez, D. Corujo, and R. L. Aguiar, "Analysis and enhancements to probabilistic caching in content-centric networking," *The Computer Journal*, vol. 58, no. 8, pp. 1842-1856, 2015. Article (CrossRef Link).

[13] E. J. O'neil, P. E. O'neil and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," in *Proc. of the 1993 ACM SIGMOD international conference on Management of data*, pp. 297-306, Jun. 1993. Article (CrossRef Link).

[14] S. Tarnoi, K. Suksomboon, W. Kumwilaisak and Y. Ji, "Performance of probabilistic caching and cache replacement policies for content-centric networks," *IEEE 39th Conference on Local Computer Networks*, pp. 99-106, Sept. 2014. Article (CrossRef Link).

[15] M. Xu, Z. Ming, C. Xia, J. Ji, D. Li and D. Wang, "SIONA: A Service and Information Oriented Network Architecture," *Journal of Network and Computer Applications*, vol. 50, pp. 80-91, 2015. Article (CrossRef Link).

[16] K. Suksomboon, S. Tarnoi, Y. Ji, M. Koibuchi, K. Fukuda, D. Abe, N. Motonori, M. Aoki, S. Urushidani and S. Yamada, "PopCache: Cache more or less based on content popularity for information-centric networking," *IEEE 38th Conference on Local Computer* Networks, pp. 236-243, Oct. 2013. Article (CrossRef Link).

[17] C. Bernardini, T. Silverston and O. Festor, "MPC: Popularity-based caching strategy for content centric networks," *IEEE International Conference on Communications*, pp. 3619-3623, Jun. 2013. Article (CrossRef Link).

[18] B. Feng, H. Zhou, S. Gao and I. You, "An exploration of cache collaboration in information-centric network," *International Journal of Communication Systems*, vol. 27, no. 9, pp. 1243–1267, 2014. Article (CrossRef Link).

[19] Y. Li, Y. Xu, T. Lin, G. Zhang, Y. Liu and S. Ci, "Self Assembly Caching with Dynamic Request Routing for Information-Centric Networking," *IEEE Global Communications Conference*, pp. 2158-2163, Dec. 2013. Article (CrossRef Link).

[20] W. Liu, S. Yu, Y. Gao and W. Wu, "Caching efficiency of information-centric networking," *IET networks*, vol. 2, no. 2, pp. 53-62, 2013. Article (CrossRef Link).

[21] Y. Xu, Y. Li, T. Lin, Z. Wang, W. Niu, H. Tang and S. Ci, "A novel cache size optimization scheme based on manifold learning in Content Centric Networking," *Journal of Network and Computer Applications*, vol. 37, pp. 273-281, 2014. Article (CrossRef Link).

[22] D. Rossi and G. Rossini, "On sizing CCN content stores by exploiting topological information," in *Proc. of IEEE Conference on Computer Communications Workshops*, pp. 280-285, Mar. 2012. Article (CrossRef Link).

[23] S. Arianfar, P. Nikander and J. Ott, “On content-centric router design and implications,” in *Proc. of the Re-Architecting the Internet Workshop*, Nov. 2010. Article (CrossRef Link).

[24] M. Varvello, D. Perino and J. Esteban, "Caesar: A content router for high speed forwarding," in *Proc. of the second edition of the ICN workshop on Information-centric networking*, pp. 73-78, Aug. 2012. Article (CrossRef Link).

[25] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Telecom ParisTech*, Technical report, 2011. Article (CrossRef Link).

[26] J. Wang, "A survey of web caching schemes for the internet," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 5, pp. 36-46, 1999. Article (CrossRef Link).

[27] A. M. K. Pathan and R. Buyya, "A taxonomy and survey of content delivery networks," *Technical Report*, 2007. Article (CrossRef Link).

[28] D. De Vleeschauwer and L. Laevens, "Performance of caching algorithms for IPTV on-demand services," *IEEE Transactions on Broadcasting*, vol. 55, no. 2, pp. 491-501, 2009. Article (CrossRef Link).

[29] S. Podlipnig and L Böszörmenyi, "A Survey of Web Cache Replacement Strategies," *ACM Computing Surveys*, vol. 35, no. 4, pp. 374-398, 2003. Article (CrossRef Link).

[30] A. Afanasyev, I. Moiseenko and L. Zhang, "ndnSIM: NDN simulator for NS-3," NDN, Technical Report NDN-0005, 2012. Article (CrossRef Link).

[31] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proc. of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 133-145, Aug. 2002. Article (CrossRef Link).

Bohao Feng received his B.S. degree from Beijing Jiaotong University in 2011. He is currently working toward his Ph.D degree in the School of Electronic and Information Engineering at Beijing Jiaotong University. His research interests are ID/Loc Split network architecture, Soft Defined Networking, Information-Centric Networking, network-based caching mechanism, Delay Tolerant Networking, mobile Internet and multicast.

Huachun Zhou received his B.S. degree from People's Police Officer University of China in 1986. He received his M.S. and Ph.D degrees from Beijing Jiaotong University in 1989 and 2009 respectively. His research interests include mobility management, mobile and secure computing, routing protocols and network management technologies. His recently research projects include Research on models and algorithms of Information-Centric mobile Internet, supported by National Natural Science Foundation of China, and Research on the future universal network, supported by National High Technology of China.

Mingchuan Zhang received his M.S. and Ph.D degrees from Harbin Engineering University and Beijing University of Posts and Telecommunications in 2005 and 2014 respectively. Currently, he works as a vice professor in Henan University of Science and Technology. His research interests include ad hoc network, Internet of Things, cognitive network and future Internet technology.

Hongke Zhang received his M.S. and Ph.D degrees in electrical and communication systems from the University of Electronic Science and Technology of China in 1988 and 1992, respectively. From September 1992 to June 1994, he was a postdoctoral research associate at Beijing Jiaotong University, and in July 1994, he became a professor there. He has published more than 100 research papers in the areas of communications, computer networks, and information theory. He is the author of eight books written in Chinese and the holder of more than 30 patents. He is now the chief scientist of a National Basic Research Program ("973" program).