Popularity-based Neighborhood Collaborative Caching for Information-Centric Networks

Xiaodong Zhu[1,2], Jinlin Wang*[,1], Lingfang Wang[1], Weining Qi[1]
[1]National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China
Email: {zhuxd, wangjl, wanglf, qiwn}@dsp.ac.cn

***Abstract*—Research on caching strategy is the key to improving network performance for Information-Centric Networks (ICNs). But it is still a great challenge to better utilize in-network caching of ICNs with low costs. In this paper, we propose a popularity-based neighborhood collaborative caching algorithm for ICNs. In the algorithm, in-network nodes track the popularity of contents, and a novel process is used for quick comparison of popularity in the algorithm. En-route and one-hop neighborhood nodes make caching decision collaboratively. Real-world topologies and different client placed scenes are used in the simulation experiments, and our algorithm performs better in terms of latency, cache hit ratio and path stretch compared with the state-of-the-art algorithms and ideal situations. The overhead and tradeoff of the algorithm on estimation of popularity and node interaction are also explored in details, and the proposed algorithm provides a practical choice for ICN caching decision strategy with its good performance and acceptable overhead.**

***Index Terms*—Information-Centric Networks, Popularity-based, Collaborative Caching**

I. INTRODUCTION

Information-Centric Network (ICN) [1] has been considered as an important way to solve the current problems of the network, and ICN treats information as the core of the network system. Many ICN architectures have been proposed, such as CCN [2], DONA [3], COMET [4] and PURSUIT [5]. And a key aspect of ICN is in-network caching. Contents can be stored in ICN nodes so that the network can respond quickly to the request. Additionally, link traffic and load on servers can also be reduced. Therefore, using in-network cache and delivering contents efficiently are important issues for ICN.

Various caching strategies are used to resolve the above issue. And these strategies can be divided into three types: individual caching, en-route caching and off-path caching. In individual caching, each node in the network individually decides whether or not to store the passing contents. This kind of strategy is simple and does not need information interaction between network nodes. But the performance achieved by this kind of strategy is low. As for en-route caching, both of the content requesting and caching happen on the path that from the consumer to the provider. Nodes along the path can work together to optimize the placement of the content. This brings a lot of benefits compared to individual caching, but the caches in off-path are not considered. In off-path caching, nodes within a certain range or all the nodes in the network are utilized to store contents collaboratively. And requests can be forwarded to the closest copy which is not on path. In [6] [7], the great performance that can be achieved from off-path is described. But complex node interaction or a content resolution server with frequent upgrades and queries is needed for each node to perceive the cache state of other nodes.

In order to better utilize the advantages of both en-route and off-path caching with low costs considering the reality, our brief design is to combine en-route caching decision with off-path nodes in a limited range. In-network nodes track the popularity of contents, and the tracking mechanism is designed based on the limited storage resources and the quick processing requirement of the node. Additionally, en-route caching is the main part to make redundant content less and more popular content near the consumers, and one-hop neighborhood nodes are considered to improve in-network cache hit and decrease the same content in the neighborhood.

The main contributions of this paper are:

1) We propose a popularity-based neighborhood collaborative caching scheme for ICN, which combines popularity-based on-path caching decision and collaborative off-path caching by using the bloom filter.
2) The popularity estimation and comparison process are optimized as the requirement of quick process for in-network nodes. There is also a detailed analysis of the overhead and statistics of cases which may influence the popularity estimation.
3) Real world internet topologies are used in the evaluations to examine the performance of algorithms. Different scenes that the way by which the client is attached to the ICN nodes are also considered. The proposed algorithm is compared to not only efficient existing on-path and off-path algorithms but also the ideal situation that requests can be forwarded to the closest copy in the network with no extra information and query delay. And our algorithm outperforms these existing algorithms and the ideal situations.

The rest of the paper is organized as follows: Section II describes the related work of the caching strategy in ICN. Then the popularity estimation and the proposed popularity-based neighborhood collaborative caching algorithm are discussed in Section III. And Section IV analyzes the evaluation performances. Finally, conclusion and future work are described in

978-1-5090-6468-7/17/$31.00

Section V.

II. RELATED WORKS

Leave copy everywhere (LCE) is the default cache decision for some ICN architectures(eg. CCN [2]). LCE is a simple strategy and just caches all the passing contents along the path. But there will be too many duplicate contents in the network and all of the contents are seen as equal without distinction. This decrease the utilization of storage within the network.

For en-route caching, caches along the path are collaboratively used. Prob-Cache [8] probably caches contents and the length of path is also considered. CL4M [9] perceives the state of topology and chooses the node with the largest betweenness centrality as the cache node. The network topology is considered in Prob-Cache and CL4M, while the popular contents still cannot be acquired quickly. Leave copy down (LCD) [10] distributes popular contents to the edge of the network by a simple mechanism that contents are stored at the next hop of the cache-hit node. But the redundancy of the content still exists and the distribution relies on many cache hit times. In age-based caching [11], popular contents have a higher probability to be cached at the edge of the network when the network is stable. But age-based caching still cannot avoid the redundant caches.

For off-path caching, a globel or local group of nodes are utilized collaboratively to unleash the advantages of in-network cache. The hash strategies proposed by [12] determine which node caching a content by hash functions. These hash strategies significantly improve the cache hit ratio, but the path length is usually large in such schemes. Mick *et al.* [13] propose multi-hop neighborhood collaborative caching to better use caches in several hops. It reduces the latency with a good cache hit ratio, while the system is a bit complex and the request forwarded between too much hops may influence the performance when used in reality. The high transmission to get the status of other cache is another problem for many off-path schemes. According to Bayhan *et al.* [14], caching the most popular contents in the name resolution server can get the most benefit. In the paper, the situation assumed is a little idealistic: popularity estimation is not considered and cache status is greatly affected by caching strategies.

As discussed above, it is an appropriate and still needs to be researched way which comprehensively considers the topology, content popularity and overhead for using in reality. In order to make more popular contents near the client, enhance in-network cache diversity, utilize off-path cache in a simple and low transmission load way, we propose a popularity-based neighborhood collaborative caching algorithm. The contents will be located hierarchically by the popularity. And the overhead for using in reality is also comprehensively considered.

III. ALGORITHM

In this section, the brief design idea about our algorithm is proposed. We first describe the popularity estimation and comparison process in the algorithm. This is the foundation of caching decision in the algorithm. Then we describe the algorithm overall. Also, the details and an example of the algorithm are shown.

A. Estimate and Compare Popularity

1) Design Principle: Comparing the popularity of the contents will be used for caching decision in our algorithm. And we consider the result is true when the popularity of compared content is not the least among the given set. So the popularity estimation is needed firstly. Considering the reality of in-network node, we should track the popularity of contents with little calculation and limited storage. Popularity comparison also needs further consideration. Obviously, it is not a suitable method to compare all the given contents every time the popularity comparison process is needed. Just compare the popularity of compared content with the least popular content in the given set is enough, but it is difficult to maintain the least popular content for quick comparison. Because the estimated popularity of the content is changing and the cached content set is also changing all the time.

Forecasting the popularity of content in traditional methods for many kinds of caching systems needs complex data structure and algorithm implementation. As the line-speed requirement of in-network caching, popularity processing of ICN nodes should be simple. Scalability is another important issue as ICN nodes' limited storage space can't keep track of evolving content popularity if they store all passing information and evict nothing. Therefore, simplicity and scalability are the main design principles for the popularity estimation and comparison process.

We use the request counts of the content to represent the popularity P_{ct} of each content in this paper. The content store of the ICN node is extended to also store the request counts of cached contents. And a LRU(Least Recently Used)-based queue named the Popularity Index Store (PIS) in each node is used to store request counts of the content which is not in the content store. There is a variable *MinContent* to index the content whose request counts is minimum among the cached contents.

2) Popularity Estimation: As shown in Algorithm 1, the situation is a request coming into a node. If there exists the required content in the content store, request counts for this content in the content store is increased by one. And if the number of counts of the content is less than that of *MinContent*, update *MinContent* with the content. If *MinContent* is exactly the required content, search the content store to find the minimum one to reset *MinContent*. This operation is necessary as the change of request counts of *MinContent* may make other content in the content store become the one with minimum counts. Additionally, just add the request counts of the requested content in PIS if it is not in the content store.

As shown in Algorithm 2, when an ICN node needs to save a content into its content store, the request counts of this content is updated with the counts stored in PIS. Next, The count index in PIS is deleted. The counts of the evicted content will be kept in the PIS instead. Similar to Algorithm 1, when

Algorithm 1 Request Process

```
in-network node get a request for content CT_a
if node has the needed content then
    add request counts of CT_a in the content store
    if CountsInCS_{CT_a} < CountsInCS_{MinContent} then
        MinContent = CT_a
    end if
    if CT_a is MinContent then
        search and reset MinContent among the cached contents
    end if
else
    add request counts of CT_a in PIS
end if
```

the evicted content is *MinContent*, search the content store to find the minimum one to reset *MinContent*. This reset action is similar to resetting *MinContent* in Algorithm 1.

Algorithm 2 Content Process

```
node put CT_a
update CountsInCS_{CT_a} with CountsInPIS_{CT_a}
delete CountsInPIS_{CT_a}
add CountsInCS_{evict} to PIS
if evict is MinContent then
    search and reset MinContent among the cached contents
end if
```

3) Overhead: Assume the cache size of an ICN node is n and using 16 bits to store the counts of each content. Each component of the content store will need extra 16 bits, which is pretty small compared to the length of the cached name and data. According to [15], the LRU-based PIS and extra store of the content store will totally need $(136 + 16)m + 16n = 152m + 16n$ bits, where m is the size of PIS. Obviously, the time complexity is $O(n)$ if we compare all contents in the popularity comparison process. We can also use the min heap and hash-map structure to maintain count indexes. And the time complexity of the operation on the structure and get the minimum one is both $O(\log n)$. While in the proposed algorithm, the structure is a simple LRU-based queue and only two situations will process the resetting *MinContent* with the time complexity $O(n)$. Other popularity comparison and operation on the data structure all only cost the time complexity $O(1)$. The resetting *MinContent* operation can also be processed by other processes to avoid the impact of line-speed packet processing in the ICN node. And the ratio of these two situations will be analyzed in details in Section IV-D.

B. Neighborhood Collaborative Caching Algorithm

1) Brief Design: The proposed algorithm is designed to cache only one copy along the path from the client to the node caching the required content to avoid redundant contents. To achieve the goal of making popular contents near the edge of the network, caching decision begins with the request coming into the network from the edge nodes. During caching decision process in each node, if the popularity comparison between the request and the cached contents of one node is true, this node is considered as the cache node for the coming back content and other node will not cache this content. If the popularity comparison is false, the next hop along the request path will do the same process. This will make more popular contents stored at the edge and less popular contents stored in the network. The cache status of one hop neighborhood nodes is also used to search the potential content location when processing requests and avoid same contents in one hop neighborhood nodes when preparing to cache a content.

2) Bloom Filter: To reduce interaction traffic, we use the bloom filter to record the cached contents. And every in-network node periodically transmits its bloom filter to the surrounding one hop neighborhood nodes. Each node stores the bloom filters got from the surrounding nodes. The cache size of each in-network node is used as the capacity m of a bloom filter. According to [16], the number of bits needed for each bloom filter is:

$$n = m\frac{\left|\ln p_{fp}\right|}{\ln^2 2}$$

3) Algorithm: A compare tag using only 1 bit needs to be stored in the request message in the proposed algorithm. And the name of the decided node for caching the content also needs to be stored in the request and the content message. If we use the MAC address of the node to indicate the node, this will only use 48 bits.

As described in Algorithm 3, the content store is checked first for a coming request. If the node has the needed content, it will respond to the request with this content. If there is no needed content in the content store, the node will check the cached surrounding bloom filters to get the potential source. Then the node will forward the ask information to confirm whether the content exists in the potential source. The ask information can be a private protocol or just the original request with a special tag for the distinction. This sending ask information process will be logged as request cost in the performance evaluations. The needed content will come back and the node will respond to the request with this content if surrounding nodes have the content. If there is no requested content in surrounding nodes, the node will check the compare tag and process popularity comparison. This node will be considered as the cache node if both checks above are true. Then the name of this node will be added to the request and the compare tag will be changed to false. And the following nodes will not do the same process as the compare tag is false. The request is forwarded to the default path at last. Additionally, the node will be decided as the cache node directly without popularity comparison if the cache is not full and compare tag is true. This is not shown in the algorithm. When the request gets the content somewhere, the node name in the request will be added to the content.

Algorithm 3 Request Process Algorithm

in-network node get a request
if the node has the needed content **then**
  respond to the request with this content
else
  search and confirm surrounding potential sources
  if surrounding nodes have the needed content **then**
    get back the needed content and respond request with it
  else
    if (compare tag == true) and (popularity comparison == True) **then**
      add name of this node to request
      change compare tag to False
    end if
    forward request to the default path
  end if
end if

Algorithm 4 Content Process Algorithm

in-network node get a content
check node's name in the content
if node's name in the content == name of the node in process **then**
  if no same content in neighborhood nodes **then**
    cache this content
  end if
end if
forward the content

Algorithm 4 shows the content process pipeline. The node in process will check its name with the node name stored in the content. The content will be stored in this node only if the name check is true and there is no same content in neighborhood nodes by searching the bloom filters.

4) Example: A simple example of caching decision is shown in Figure. 1. Each node can store one content and the popularity rate is $A > B > C > D > E$ in this example. And the current cache status is also shown in Figure. 1. The client sends a request for content B into the network. There is no change to the request as the popularity of B is lower than A at R3. Then the request is forwarded to R2. There is no content B in R2 or one-hop neighborhood nodes of R2. As the popularity of B is higher than C which is stored at R2, R2 changes the compare tag of the request to false and adds its name to the request. The request is forwarded to R1 after the above process, and there is no cache hit at R1. There is no cache decision at R1 as the compare tag has been changed to false. The request is finally forwarded to the server to get the content B. And content B will be stored at R2 when it comes back to the client with the name of R2 stored in the content B.

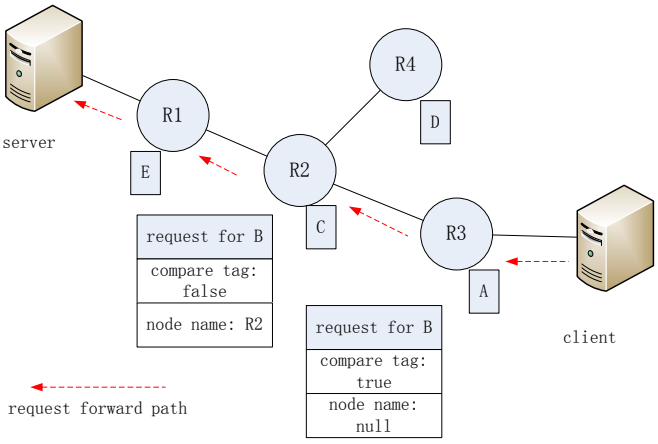


Fig. 1: An example of caching decision

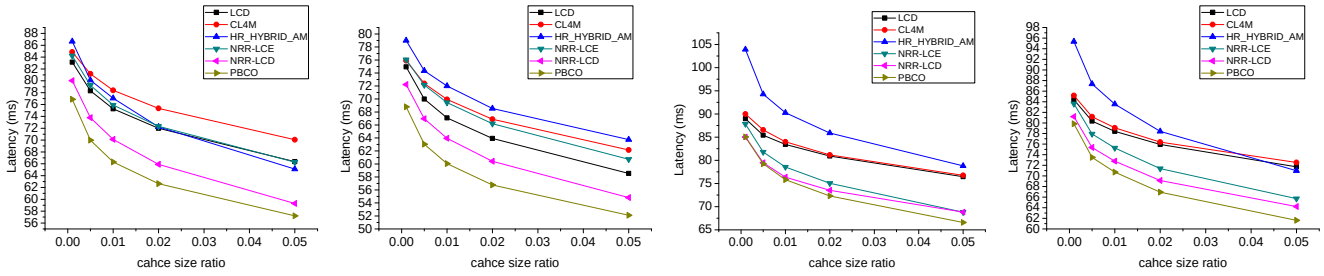
IV. PERFORMANCE EVALUATIONS

A. Simulation Setup

We implement our algorithm based on Icarus [17], a python emulator for ICN caching strategy simulation. We modify the strategy process pipeline in the simulation to implement the popularity index, bloom filter transmission and caching decision in the proposed algorithm. Through the experiment, we conduct a detailed performance analysis and cost statistics.

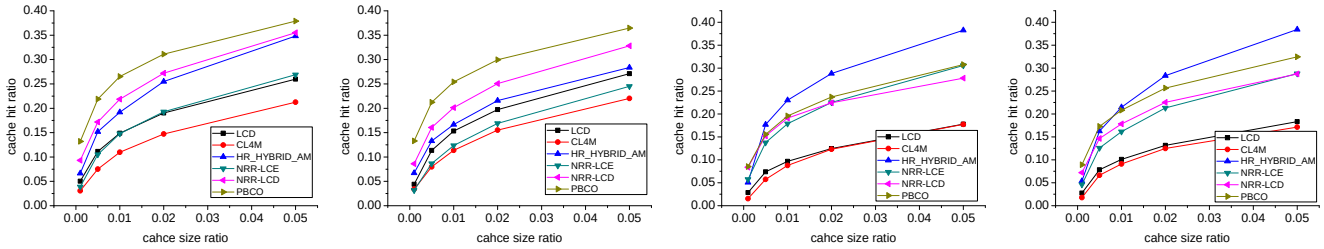
Three metrics are used to measure the performance of the algorithms. They are latency, cache hit ratio and path stretch. Latency reflects the delay in getting content, which is the intuitive metric to effect the network performance. Cache hit ratio measures the portion of content requests served by a cache. And a higher cache hit ratio means that the response time and the load of a server is less. Path stretch means the ratio between the actual path length with the calculated shortest path. This reflects the transmission within the network from another aspect. In addition, work load metric is not presented due to the restriction of the paper space. But it is utilized to calculate the transmission overhead of the bloom filter.

The evaluation is performed with two real word topologies: GEANT (European academic network), WIDE (Japanese academic network) and two Rocket Fuel topologies: AS1221 (Telstra, Australia) and AS1775 (EBONE, Europe). And in GEANT and WIDE, only nodes with degree equaling to 1 are set as clients. While in AS1221 and AS1775, each of the nodes in the network is linked with a client. This can better reflect the possible different ICN situations. We set the number of contents as $N = 10^5$. This size can be considered fair to simulate a realistic scene, which was analyzed in [18]. The popularity of the contents is defined by Zipf distribution, and the impact of different values of parameter α on the experiments is also tested. We assume that the requests of contents are generated as $\lambda = 10$ per second following a Poisson process. The cache of each node is empty at the beginning, and cache size of each node is equal. The first



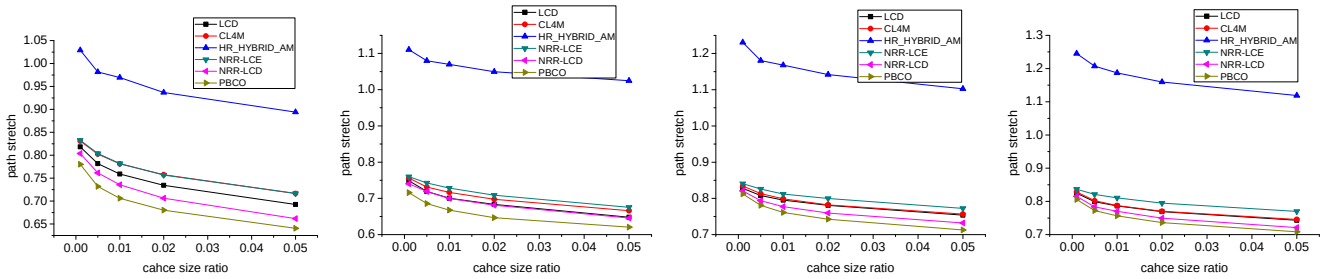
(a) GEANT (b) WIDE (c) AS1221 (d) AS1755

Fig. 2: Latency with different cache size ratio



(a) GEANT (b) WIDE (c) AS1221 (d) AS1755

Fig. 3: Cache hit ratio with different cache size ratio



(a) GEANT (b) WIDE (c) AS1221 (d) AS1755

Fig. 4: Path stretch with different cache size ratio

$T = 3$ hours are used to warm up caches in the network and the next $T = 12$ hours are logged to get statistics. We set that the size of the PIS is 5 times the size of the content store and the false positive probability value $p_{fp} = 0.05$ is selected in the proposed algorithm. Least Recently Used (LRU) is used as the cache replacement strategy in the evaluation, which is the most widely used strategy in ICN. And cache size ratio is the ratio of the number of contents that each node can cache to the total number of contents. In addition, α is 0.8 and cache size ratio is 0.01 in the evaluations if not specified.

The proposed algorithm is compared to some traditional and state-of-the-art algorithms: Leave Copy Down (LCD) [10], Betweenness Centrality Cache Less For More (CL4M) [9] and Hybrid Asymmetric-Multicast Hash Routing (HR Hybrid-AM) presented in [12]. This hash routing scheme has good performance with little affected by topology changing. Additionally, we also consider the ideal situation that each node knows the closest node having the requested content without extra information. Therefore, a node can directly send the request to the nearest content location. And the caching strategy for this situation is Leave Cope Everywhere (LCE) [2] and LCD. They are identified as NRR-LCE and NRR-LCD in the evaluation. This situation is ideal as a content resolution system which knows the status of all the temporary cache is needed. It is difficult to realize because the rapid changes in the network cache and the update cost and delay. The cost and delay to request the closest copy are also ignored in the situation. But it is a very good reference benchmark which shows the best promotion which can be got from the off-path. And our algorithm is identified as PBCO for short.

B. *Effect of Cache Size Ratio*

We first show the effect of different cache size ratio for all four topologies. The parameter α of Zipf distribution is set as 0.8 here. Fig. 2, 3, 4 show the performance of latency, cache hit ratio and path stretch. From these figures, we can observe

that: First, the proposed algorithm outperforms than all other algorithms in almost all cases. And the performance is stable with the growth of the cache size ratio. Second, in Fig. 3(c) and Fig. 3(d), hash strategy performs better when the cache size is large. As its design to locate contents by hash, larger cache size in the network can cache more diverse contents. And as each node has a client in AS1221 and AS1775, the diversity of PBCO is less than HR Hybrid-AM with a large cache size. And some cache nodes are not clients in GEANT and WIDE, so PBCO has more chance to locate different contents along the path. So PBCO is better than HR Hybrid-AM in Fig. 3(a) and Fig. 3(b). And hash strategy sacrifices the performance of latency and path stretch, and the gap between hash strategy and PBCO is pretty large. Additionally, PBCO also has better cache hit ratio than HR Hybrid-AM when the cache size is small in Fig. 3(c) and Fig. 3(d). Finally, the ideal situation NRR-LCD also performs well as it can get the nearest replica in the network. NRR-LCD is the best except for PBCO. But the proposed strategy still outperforms NRR-LCD.

C. Effect of Workload's Popularity

In this section, the α of Zipf distribution is changed from 0.6 to 1.4 with fixed cache size ratio (0.01) to analyze the effect of workload's popularity. As shown in Fig. 5, 6, 7: First, PBCO performs well with the increment of α and is still the best in almost all cases. Second, the performance of all strategies gains a lot with the growth of α. And the gap between PBCO with compared strategies gets smaller when the α is large. When the α is large, only some of the most popular contents are requested. Any caching strategy can identify contents with high popularity with this skewed enough popularity. Generally, the α is no greater than 1 in reality [19]. Finally, HR Hybrid-AM performs better on cache hit ratio in Fig. 6(c) and Fig. 6(d), and PBCO is better in Fig. 6(a) and Fig. 6(b). There are more content types of HR Hybrid-AM whose design that finds content accurately is the advantage with the large α. Topology reason analyzed in Section IV-B is the same here.

D. Overhead

Storage overhead: As analyzed in Section III-A3, the total storage needed for indexing popularity is $152m + 16n$ bits, where m is the size of PIS and n is the cache size. As $m = 5n$ in the simulation, $776n$ bits are used for each node. At the situation that each chunk size is $10K$ [20], consumption of space storage for indexing popularity is $\frac{776}{8\times 10K} \approx 0.95\%$ of the space to store a content. And in our simulations, with the biggest cache size $10^5 \times 0.05 = 5000$, each node only needs $\frac{5000\times 776}{8\times 1024} \approx 474KB$. And each bloom filter is $3.8KB$ as described in Section III-B2 with 5000 contents and 0.05 error rate. Each ICN node also needs to store the one-hop surrounding nodes' bloom filter, and the number of the one-hop surrounding nodes is usually a single digit.

Calculation overhead: As analyzed in Section III-A3, the popularity comparison process that may influence the line-speed process in the pipeline of our algorithm has been solved by using *MinContent* to compare with time complexity $O(1)$. And now the main calculation overhead is resetting the *MinContent* with time complexity $O(n)$. There are two cases that *MinContent* needs to be re-calculated. One is that the request counts changing content is *MinContent*, and another is that the evicted content is *MinContent*. We define the former as case 1 and the latter as case 2 for short. Table. I shows the counts of each case during the evaluation. The total ratio is very low and the number of case 1 drops with the growth of the cache size ratio. The larger the cache size ratio is, the less times request counts of *MinContent* changes. This is also consistent with the law of Zipf distribution. While case 2 is the most frequent at 0.01 cache size ratio with the influence of both cache size and cache put action. We compare the time consumption directly with no considering the advantage that resetting *MinContent* can be done without in the ICN pipeline. Assume c is the total request counts change times, d is both two cases happen times and n is the cache size. The time consumption between the proposed process and min heap is:

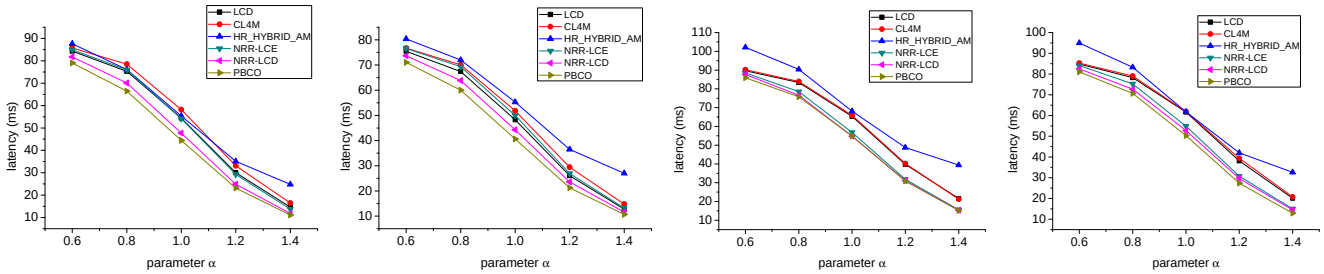
$$\frac{dO(n) + cO(1)}{cO(\log n)} \approx \frac{dO(n)}{cO(\log n)}$$

And d/c is the total ratio in the Table. I. The lowest value of the result is 0.013 and up to 0.3 of all situations.

TABLE I: Get *MinContent* statistics

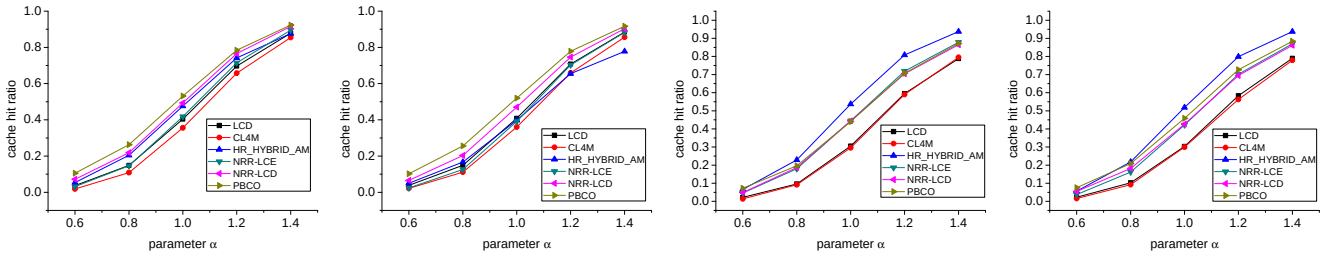
Topology	Cache size	Counts Change	Case 1	Case 2	Total ratio
GEANT	0.001	1433016	1748	260	0.00140124
	0.005	1350161	418	373	0.000585856
	0.01	1306000	239	455	0.000531394
	0.02	1259751	146	433	0.000459615
	0.05	1191711	66	338	0.000339008
WIDE	0.001	1022480	831	58	0.000869455
	0.005	974960	259	261	0.000533355
	0.01	946768	178	308	0.000513325
	0.02	917845	95	312	0.00044343
	0.05	874353	53	203	0.000292788
ASN1221	0.001	1968555	7849	645	0.00431484
	0.005	1881982	1907	2484	0.002333179
	0.01	1831272	939	2714	0.001994788
	0.02	1776931	451	2015	0.001387786
	0.05	1688447	188	1285	0.000872399
ASN1775	0.001	1878773	5678	341	0.003203687
	0.005	1786271	1122	1904	0.001694032
	0.01	1744022	669	2320	0.001713855
	0.02	1690354	343	1741	0.001232878
	0.05	1610402	163	1032	0.000742051

Bloom filter overhead: The false positive ratio and the bloom filter transmission ratio are shown in Table. II. The false positive ratio shows the ratio of ask information getting false at the potential source node to the total counts of sending ask information. This may happen as the false positive characteristic of the bloom filter or the cache being evicted during the transmission gap. The false positive ratio is higher in AS1221 and AS1775, because each node is linked with a client in these topology and cache changes more frequently. When the cache size is large, cache changes less frequently and the false positive ratio declines. The transmission ratio shows the ratio of total bloom filter transmission and the total link load. The transmission ratio increases with the cache size as the size of bloom filter increases and the total link load decreases. The transmission ratio in AS1221 and AS1775 is



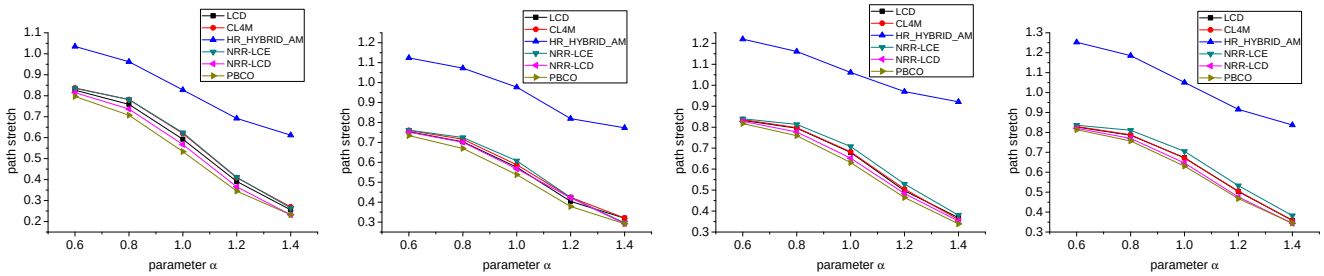
(a) GEANT (b) WIDE (c) AS1221 (d) AS1755

Fig. 5: Latency with different parameter α



(a) GEANT (b) WIDE (c) AS1221 (d) AS1755

Fig. 6: Cache hit ratio with different parameter α



(a) GEANT (b) WIDE (c) AS1221 (d) AS1755

Fig. 7: Path stretch with different parameter α

also larger than that in GEANT and WIDE. The reason is similar, each node is a client in AS1221 and AS1775, and the average path length is smaller compared to GEANT and WIDE. As request in GEANT and WIDE may have to pass through many intermediate nodes. So the link load caused by each node is less in AS1221 and AS1775. This results the higher transmission ratio.

E. Discussion

The proposed algorithm provides a practical choice for ICN caching decision strategy to use in reality with its good performance and acceptable overhead. Additionally, The brief design principle considers being used in reality as the most important thing. The popularity estimation is simple here because of such principle. Indeed, the popularity of content can be forecasted more accurately with a complex system to adapt to different scenes. But it needs to balance the improved performance and the cost. This is also a research direction worthy of consideration in the future. The evaluation results show the great performance the proposed algorithm can get. While there are still differences of results between different topologies and cache size ratio. And we think that the situation that limited cache size ratio and only a part of the ICN nodes being attached to client, is more close to the real.

V. CONCLUSION

In this paper, a popularity-based neighborhood collaborative caching algorithm for ICN is proposed to better utilize in-network caching of ICN. And the popularity estimation is optimized for quick popularity comparison in the algorithm. The evaluation results show that the proposed algorithm is better than both the sate-of-the-art algorithms and ideal situations on latency, cache hit ratio and path stretch. Moreover, overhead analysis is shown for the possible deployment in reality in the future.

TABLE II: Bloom filter statistics

Topology	Cache size	false positive	transmission ratio
GEANT	0.001	0.017048699	0.00020588
	0.005	0.018217565	0.001099729
	0.01	0.019947411	0.002286292
	0.02	0.028523958	0.00475654
	0.05	0.017462384	0.012627577
WIDE	0.001	0.012572695	0.000184611
	0.005	0.022668569	0.000969304
	0.01	0.020788684	0.001998073
	0.02	0.021464023	0.004101039
	0.05	0.018854617	0.010797986
ASN1221	0.001	0.085490004	0.000937336
	0.005	0.074529822	0.004894195
	0.01	0.056858621	0.010039883
	0.02	0.036220493	0.020607586
	0.05	0.018428454	0.053670723
ASN1775	0.001	0.050156111	0.000977008
	0.005	0.044512413	0.005110064
	0.01	0.035636928	0.010458639
	0.02	0.023279465	0.021504309
	0.05	0.013821846	0.055819811

The proposed algorithm can be further optimized to estimate the popularity under a low overhead and be used in some real systems, such as Fog Computing [21] and Sea Computing [22].

Acknowledgment

This work was supported by "The Next-Generation Broadband Wireless Mobile Communications Network" National Science and Technology of Major Projects (No. 2017ZX03001019).

References

[1] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[2] 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*. ACM, 2009, pp. 1–12.

[3] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proceedings of ACM SIGCOMM 2007: Conference on Computer Communications*. ACM, 2007, pp. 181–192.

[4] G. García, A. Beben, F. J. Ramón, A. Maeso, I. Psaras, G. Pavlou, N. Wang, J. Śliwiński, S. Spirou, S. Soursos *et al.*, "Comet: Content mediator architecture for content-aware networks," in *Proceedings of Future Network & Mobile Summit (FutureNetw), 2011*. IEEE, 2011, pp. 1–8.

[5] D. Trossen and G. Parisis, "Designing and realizing an information-centric internet," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 60–67, 2012.

[6] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-centric networking: seeing the forest for the trees," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011, pp. 1–1.

[7] M. Dräxler and H. Karl, "Efficiency of on-path and off-path caching strategies in information centric networks," in *Proceedings - 2012 IEEE Int. Conf. on Green Computing and Communications, GreenCom 2012, Conf. on Internet of Things, iThings 2012 and Conf. on Cyber, Physical and Social Computing, CPSCom 2012*. IEEE, 2012, pp. 581–587.

[8] 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.

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

[10] 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.

[11] Z. Ming, M. Xu, and D. Wang, "Age-based cooperative caching in information-centric networking," in *Proceedings of International Conference on Computer Communications and Networks, ICCCN*. IEEE, 2014, pp. 1–8.

[12] 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*. ACM, 2013, pp. 27–32.

[13] T. Mick, R. Tourani, and S. Misra, "Muncc: Multi-hop neighborhood collaborative caching in information centric networks." in *Proceedings of the 2016 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 93–101.

[14] S. Bayhan, L. Wang, J. Ott, J. Kangasharju, A. Sathiaseelan, and J. Crowcroft, "On content indexing for off-path caching in information-centric networks." in *Proceedings of the 2016 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 102–111.

[15] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*. ACM, 2011, pp. 44–49.

[16] P. S. Almeida, C. Baquero, N. Preguiça, and D. Hutchison, "Scalable bloom filters," *Information Processing Letters*, vol. 101, no. 6, pp. 255–261, 2007.

[17] L. Saino, I. Psaras, and G. Pavlou, "Icarus: a caching simulator for information centric networking (icn)," in *Proceedings of the 7th International ICST conference on Simulation Tools and Techniques*. ICST, 2014, pp. 66–75.

[18] V. Sourlas, I. Psaras, L. Saino, and G. Pavlou, "Efficient hash-routing and domain clustering techniques for information-centric networks," *Computer Networks*, vol. 103, pp. 67–83, 2016.

[19] M. Yamamoto, "A survey of caching networks in content oriented networks," *IEICE Transactions on Communications*, vol. 99, no. 5, pp. 961–973, 2016.

[20] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, 2011.

[21] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Proceedings of Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, 2014, pp. 169–186.

[22] T. Jing, "Sea-cloud coordinative and look ahead (preface)," *Journal of Network New Media*, vol. 3, no. 1, pp. 1–1, 2014.