Routing and Caching Strategy in Information-Centric Network (ICN)

Liangyu Tan
Department of Computer Science,
Iowa State University

Abstract

The main usage of Internet today is content distribution and retrieval. In today's Internet, connections and data exchanging can only happen between hosts, which is also called host centric end-to-end communication. As the network users and demand of contents grows quickly, current network paradigm is getting more and more complicated and can barely meet the needs in the future. Recently, the architecture of information/content centric networking (ICN) has been proposed and is expected to replace the current communication model. As an in-network caching system, the cache management scheme is a key factor of ICN. To improve the performance of this architecture, a lot of effort has been put by many researchers into this area. In this paper, a new strategy about content caching and routing is introduced. And the results of this new scheme show that this strategy leads to a good performance, i.e., the new scheme can reach less hops, when comparing with regular LRU cache strategy. And less hops means the requested object can be found in nearer routers, the network traffic, hence, is reduced.

1 Introduction

As we already know, the Internet was designed in 60s-70s last century, and the role it played in people's daily life is becoming more and more important. Even though the user numbers of Internet increased from hundreds or thousands at the very beginning to billions by now, the Internet paradigm, however, does not change much. The sender-driven end-to-end based model still plays a dominant role. As the Internet user number grows along the time, which subsequently cause the growth of Internet traffic. And corresponding techniques, like Peer-to-Peer (P2P) and Content-Delivery Networking (CDN) are developed to meet the increasing need of Internet. And the result is that Internet becomes more and more complex and the host-based TCP/IP Internet is becoming too heavy to offer the best performance to the end-users in the near future.

Internet traffic keeps growing in the past thirty years, as computers and mobile devices getting cheaper and more affordable to public. According to Cisco's VNI forecast [1], by the end of 2022 there will be 4.8 billion global internet users and 28.5 billion network devices and connections. Most of the traffic could be attributed to content retrieval applications. And the forecast shows that up to 82 percent by 2022, which is 59 percent in 2017, of the traffic will be video related. And this increase is mainly caused by the increasing demand of video contents, like User-Generated Content, time-shift TV and high definition video on demand (VoD). And it is very likely that this growing will continue in the future.

To better handle the increasing Internet traffic, Information-Centric Networking (ICN), which has a receiver-driven content retrieval paradigm, is considered as an excellent alternative for current Internet diagram. And a handful of research communities are motivated to develop new ICN architectures. Among all the works, four architectures are now widely accepted and developed [2, 3]. They are Data-Oriented Network Architecture (DONA) [2, 5, 18], Content-Centric Networking (CCN) [2, 5, 7], Publish-Subscribe Internet Routing Paradigm (PSIRP) [2, 5, 17, 23], and Network of Information (NetInf) [2, 5, 21]. Even though there are differences on the details of those architectures, there share many assumptions, objectives, and architecture properties. One most common idea among them is to develop a network architecture which can access and distribute content efficiently, since content communication is the prevailing usage of networks in current and future.

Compared with current Internet paradigm, ICN has following features. First, data is split into small chunks/objects, usually they are called Named Data Object (NDOs) [3, 4]. NDO is independent of locations, storage method, application program and transportation methods. Second, each NDO has a globally unique name. Names are used for identifying objects independent of its location or container. In other words, if two NDOs have the same name then they are for all purpose equivalent. Also, unified name simplified the security checking of content as the names are self-certifiable, which means each content name is signed by its legal publisher and is binded with the related content. Third, in current network routers are only used for data forwarding, routers in ICN, however, have limited space to store NDOs. Combine with the unified name feature, ICN makes caching a general, open, and transparent service, independent of application.

Basically, routing strategy describes the way how an income request is forwarded, and caching strategy is about when and how an object should be cached when a new object, which is not included in the current node, arrives a node. In this report, a new caching and routing strategy is proposed for ICN, such that the popular items will ‘pop up’ to the edge routers and consequently reduces the average hop number for all data retrieve.

The rest of the report is structed as follows. Section 2 shows related work on ICN, i.e., cache management. Section 3 describes the model used in this report for simulation. Sections 4 gives detailed information about the routing and caching strategy proposed in this report. The results are showed and analyzed in section 5 and section 6 concludes this report.

2 Related work

Unlike the traditional cache systems that are closed and application-dependent designed for one particular traffic class, the cache in ICN is transparent. And this is the result of two facts. First, all contents in ICN are named under the same naming scheme, i.e., have a same name pattern. Second, ICN makes its routing and caching decisions on unified content names, and essentially making these names network-aware. In other words, caching in ICN is a general, open and transparent service, independent of applications.

And a consequence caused by this feature is that different types of traffic/applications have to contend with and share the storage in each single cache node since the cache space is usually limited. And from this aspect, cache management is a key factor that influences the performance of ICN.

In most ICN, the default cache policy is Leave Copy Everywhere (LCE) [2, 4, 6, 9, 19]. To be more specific, the target object will leave a copy on each node along the downloading path. And the obvious disadvantage of this strategy is that it introduces high cache redundancy, i.e., the same object is unnecessarily copied at multiple nodes. And this brings the result that the content diversity of the whole network is reduced. Since the total space is limited and too much copies of same content are included. To make the ICN has a better performance and reduce the redundancy caused by LCE, a lot of work has been devoted to ICN caching topic and quite a few schemes are proposed.

The first one is Leave Copy Down (LCD) [2, 9, 11, 12]. In this scheme, when a target object is found, a copy of this target will be cached only on the direct downstream node, and this avoids a large number of copies of the same object. Also, this strategy implies that an object can be pulled down to the edge sever only when it is requested many times. The second scheme is Move Copy Down (MCD) [2, 11, 12]. In this scheme, when a cache hit happens, a copy of the target will be left on the downstream node and the target on hit node will be deleted. Compared with LCD, MCD has even less redundancy. The third scheme is Copy with Probability (Prob) [2, 11]. When a cache hit occurs, each node along the returning path has a given probability p to cache a copy of the target. When p = 1, this scheme is LCE. The forth one is named Randomly Copy One (RCOne) [2, 13]. This scheme copies the requested object at one random node along the returning path. The fifth one is Probabilistic Cache (ProbCache) [2, 14]. In this scheme, when a cache hit occurs, each node along the returning path has a probability to cache a copy of the target. The difference between this scheme and Prob is that the probability to cache a copy of each node along the path is different, but not a constant. Typically, the probability is inversely proportional to the distance from the requester to the node. In other words, a node closer to the requester has a higher probability caching a copy of the object. The advantage of this scheme is that an object can be easily pushed to the network edge without creating much redundancy. And Figure 1 is used to illustrate the ideas of them.

Besides the caching strategy, replacement strategy or eviction strategy is another important aspect of cache management. Random replacement (RR) is the least complex of the basic cache replacement policy [6, 8]. This scheme simply evicts a random data chunk every time it needs to cache a chunk and the space is full. Since there is no much logic behind this policy, it is usually used as a benchmark for evaluating other replacement policies. A widely used strategy is Least Recently Used (LRU) [6, 8]. In this policy, the least recently used chunk will be evicted when the space is full, and a new chunk must be cached. The logic behind this one is that unpopular or outdated data will be more likely to be removed and keep the freshest and most popular data. And this strategy proved to be effective in practice. Another popular eviction strategy is Least

Frequently Used (LFU) [6, 8], which is a variation of LRU. The main idea of LFU is that the system keeps the record of how many times each item in a node is requested and evict the one with least requested number when the node is full, and a new data must be cached. In other words, the most popular items are kept. But the disadvantage of this scheme is that if the popular items changes frequently in a short time, then the performance of the network is poor. Because this will increase the traffic, and finally result in higher origin server load. A replacement policy named Max Diversity Most Recent (MDMR) is proposed in [6, 8]. As the name implies, this one aims to maintain the maximum diversity of the contents stored in a node. In this policy the name of a chunk is assumed to contain the information of its producer. When a new content chunk is to replace an old one, MDMR first tries to replace the oldest chunk from the same producer. If no such kind chunk exists, MDMR simply replace the oldest one in the node.

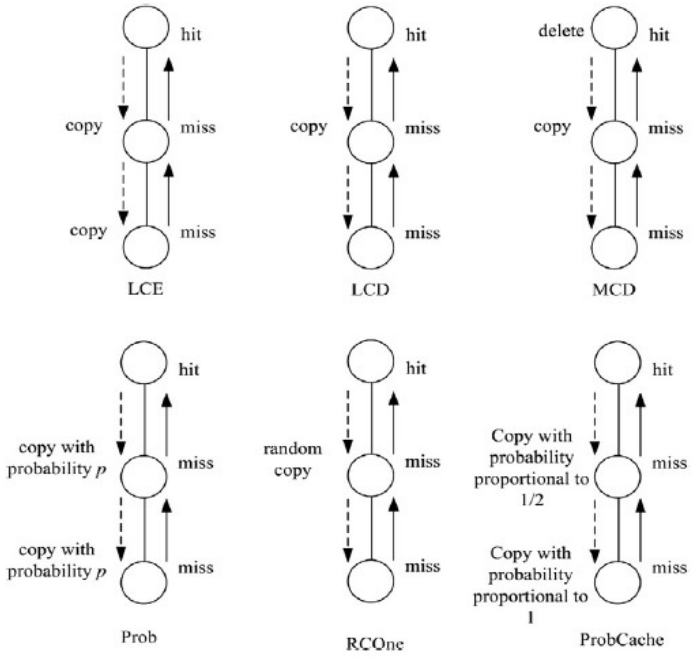

Figure 1*. Illustration of different cache schemes.

Besides the aforementioned cache management schemes, there are also other excellent ideas proposed. In [20] and [22], the authors suggest that rather than simply remove the object to be evicted, the object, instead, can be pushed back to one-level upstream, possibly incurring a cascading of object replacement. To improve the performance of the network, authors in [25] and [26] proposed the idea of implementing cache cooperation by jointly considering the content placement and dynamic request routing. In [24], [27] and [28], the idea that routing towards nearest replicas, rather than towards nearest designated servers is proposed. Also, the author in [24] argues that the content should only be cached at edge servers. But the authors in [10] do not think relying solely on the storage and processing capacities of edge cache servers along would be sufficient to scalability and performance of future ICN. Especially, with the fact that the need of

* Cites from [3]

various types of content or information grows explosively. To correct this problem, [10] proposed the idea of 'Big Cache'. In which, a path from one edge serve to the origin server is considered as one big cache unit. And this propose partly addressed the problem of cascade thrashing in a hierarchical network.

Based on so much previous work, a new cache management scheme is proposed in this report and detailed in the next following two sections.

3 System Model

In this model, there is only one origin server. The origin server is the server that contains all the contents, and all the requests will finally be routed to it if the requested objects cannot be found in routers along the path to it. And each router usually has two neighbors, which are called left neighbor and right neighbor, one parent and some children. For each router, its neighbors can be null, i.e., no neighbor, and has no child, but it must have a parent (for some routers, their parent is origin server). The routers that located on the edge of the network are called edge router/server, and every income request first reach one edge router when it come into the network, and then routed according to the routing policy described in the next part if the object is not found in this edge router.

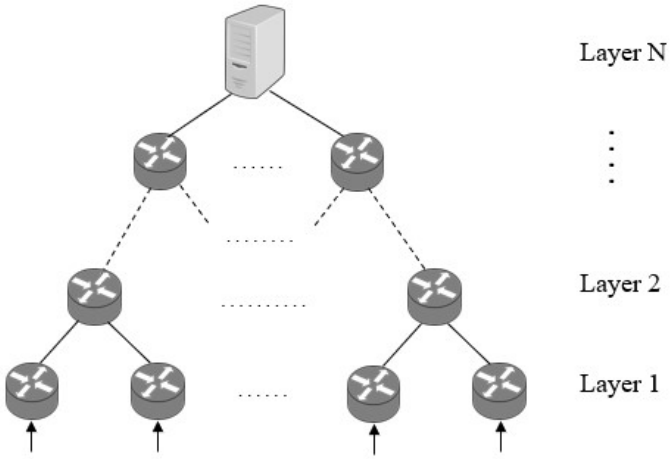

Figure 2. ICN Network model.

And the structure of the whole model is as shown in Figure 2. The whole structure can be considered as composed by multiple layers. Layer 1 are composed by edge routers which has not children. And in a typical network, layer 1 (edge routers) has the largest number. And layer 2 is the layer above layer 1. In other words, the parent of nodes in layer 1 are all in layer 2. And layer 2 usually has less nodes than layer 1. Similarly, layer 3 is on the top of layer 2 and has less nodes than layer 2, and so on so forth. The last layer, which we call it layer *N*, contains only one node, which is the origin server. And layer *N* has no parent since it is the top layer. One thing needs to be noted is that the real structure of a

network can be different with the model shown in Figure 2 in many ways. For example, a node can have more than two children on the downstream level, and the nodes number of one level can be close to its downstream layer or way much less then it. But all those changes do not affect the performance of the scheme proposed in this report.

The Zipf distribution is commonly used in traditional ICN research [15], because web content generally follows this pattern. Zipf's law states that given a large sample of objects used, the frequency of any object is inversely proportional to its rank in the frequency table. For example, the most frequent object will occur about twice as often as the second most frequent object, three times as often as the third most frequent object, and so on. And Figure 3 shows the probability distribution of a thousand objects.

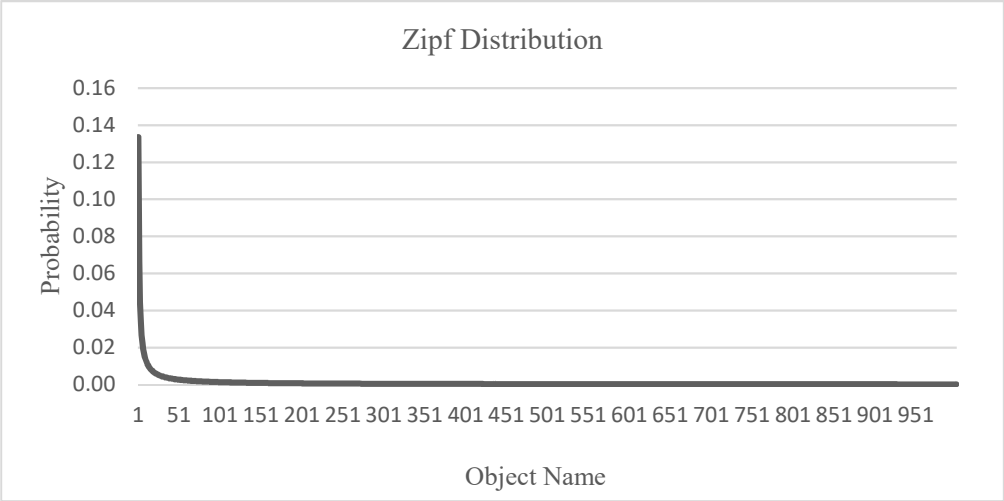

Figure 3. Probability distribution of 1000 objects that follows the Zipf's Law.

4 Caching and Routing Strategy

In this part three algorithm will be introduced. The first one is the most basic one proposed, the second one is improved based on the first one and has better performance than first one, and the last one is the final proposition of this paper and has the best performance. This part will introduce the details of each algorithm, the performance of them will be presented and analyzed in Section 5. In the rest of this section, *O* denotes the requested object, *N* is the current node that *O* reaches, *Item* stands for the list in each router that stores objects, and *C* is the storage capacity of a router.

4.1 Basic Caching and Routing strategy

The idea behind this strategy is that each router is considered as a unit. When a request reaches this router, if the requested object is stored in it, then the object will be returned to the router where the request comes from. And the requested item will be moved to the head of the list which stores all the objects. If the item is not stored in it, then the request will be passed to its parent, and so on so forth. When the object is returned, there are two cases. Case 1, the size of list *Item* is smaller than *C*, in other words, there are still space available, then this object can be added to the *Item* directly. Case 2, *Item* is full, then the

last object in this list *Item* be removed and the returned object will be added to the first place of the *Item*. In this way, *Item* contains only the most recently requested objects.

4.1.1 Caching

Algorithm of Caching:

if Item in Node N contains object O
    move O to the first position of Item
else if Node N does not contain object O
    if Item.size < C
        Item add O
    else
        remove the last object in Item and add O to the first position of Item

4.1.2 Routing

Algorithm of routing on income request.

if Item contains object O
    return O to the router where the request comes from
else if Item does not contain object O
    route the request to its parent
cache O as descripted in caching algorithm when O is returned
return O to the router where the request comes from

In this routing policy, if object *O* is stored in current router, then *O* will be responded directly. Otherwise, current router will route this request to its parent and return *O* to the router where the request comes from once *O* is returned from its parent.

4.2 Big Cache Caching and Routing strategy

The idea behind this strategy is that each path (from edge router to origin server) is considered as a big cache unit. When a request reaches this router, if the requested object is stored in it, then the object will be returned to the router where the request comes from. And the counter of the requested object will increase one and the *Item* will be resorted according to the descending order of the counter of each object stored in the list. If this object happens to be the first object in the *Item* of current node, i.e., the most popular object in the list, and if this router is not edge router, then the counter of this object (name it $O_{current}$) will be compared with the counter of the last object (name it $O_{previous}$in the *Item* of the router where the request comes from), i.e., the least popular object in the list . If the counter of $O_{current}$ is greater than the counter of $O_{previous}$, then those two items will be swapped. In this way, the popular objects 'pop down' to the edge routers. If the object is not stored in current node, then the request will be passed to its parent, and so on so forth. When the object is returned from current router's parent, the object will be passed to the router where the request comes from.

4.2.1 Caching

Algorithm of Caching:

if Item in Node N contains object O
    Get the index i of object O in Item
    Item[i].count = Item[i].count + 1
    if i == 0 && Item[i].count in N > Item[C – 1].count in P
        Swap Item[i] in N > Item[C – 1] in P
    else if Item[i].count == Item[i - 1].count
        Swap Item[i] and Item[i – 1]
else if Node N does not contain object O
    if Item.size < C
        Item add O
    else
        continue

In addition, if there are two content has the same requested number, then the most recent requested item will be placed ahead. when the list size in the router is less than *C*, in other words, the list is not full, then it stores all the contents it delivered. Also, there is a special case, when the parent of current router is the origin server, the newly requested object from origin server will always replace the least requested object in current node's *Item* when the *Item* is full.

4.2.2 Routing

Algorithm of routing on income request.

if Item contains object O
    return O to the router where the request comes from
else if Item does not contain object O
    route the request to its parent
        cache O as descripted in caching algorithm when O is returned
        return O to the router where the request comes from

Each request sent out from a router will remember its coming path, once the target is found the content will be delivered along the path that the request came along.

4.3 Improved Big Cache Caching and Routing strategy

First, the Big Cache idea is same as mentioned in Section 4.2. The difference is that in this strategy a router and its left and right neighbors (if exist), rather than just itself, is considered as a big cache unit (as shown in Figure 3).

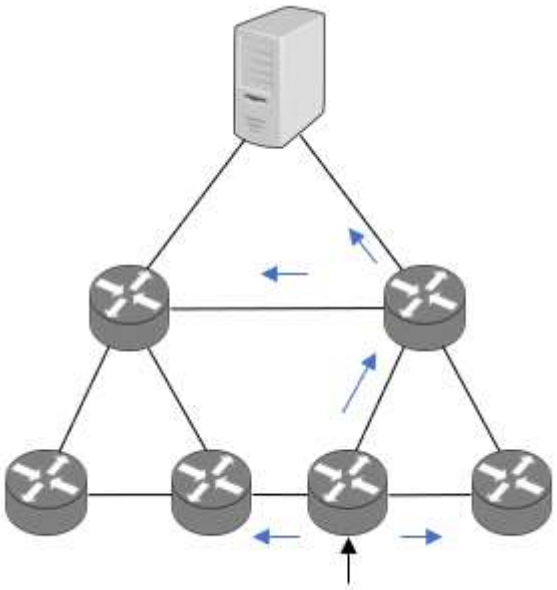

Figure 3. A router and its left and right neighbors compose a cache unit.

When a request reaches a router, if the requested object is stored in it, then the object will be returned to the router where the request comes from. And the counter of the requested item will increase one and the list will be resorted according to the descending order of the counter of each item stored in the list. If this object happens to be the first object in the list, i.e., the most requested object in the list, and if this router is not edge router, then the counter of this object (name it $O_{current}$) will be compared with the counter of the last object (name it $O_{previous}$) in the list of the router where the request comes from, i.e., the least popular object in *Item*. If the counter of $O_{current}$ is greater than the counter of $O_{previous}$, then those two items will be swapped. If the requested object is not stored in it, then the request will firstly be passed to its neighbor. If the requested object is not found after the searching on both neighbors, then the request will be passed to the next level. And the search in the next level will follow the same search strategy as in current level.

4.3.1 Caching

Algorithm of Caching: this algorithm is exactly same as in part 4.2.1.

4.3.2 Routing

The difference between the routing strategy in 4.2 and 4.3 is not much. In 4.2, the request is only routed to its parent if the requested object is not found in current router. In 4.3, the request will be routed to its neighbors first if current node do not have the requested object, and then the request will be routed to its parent if the requested object is not found in its left and right neighbors. The performance, however, as shown in the next part is improved since downstream layers contains more popular objects than upstream layers.

Algorithm of routing on income request.

if Item contains object O

return O to the router where the request comes from

else if Item does not contain object O

route the request to its right neighbor (if it as one)

if O is found

cache O as descripted in caching algorithm when O is returned

return O to the router where the request comes from

return

if O is not found on first right neighbor, keep routing the request (only) to right routers on the same level till found the object or reaches K_{th} neighbor

route the request to its left neighbor (if it as one)

if O is found

cache O as descripted in caching algorithm when O is returned

return O to the router where the request comes from

return

if O is not found on first left neighbor, keep routing the request (only) to left routers on the same level till found the object or reaches K_{th} neighbor

route the request to its parent

keep the same level order routing strategy as described above

return O to the router where the request comes from when it is found

5 Results and Analysis

In this part, we will show the results from three different algorithms and run them in networks with different router capacities and topologies. To test the performance of the algorithms, 1000000 requests will be made. There are total 1000 objects stored in the origin server. And probabilities of the objects follow the Zipf's distribution as described in Section3. There are three different typologies as shown in Figure 4. In the simulations, the capacity of the origin server is 1000, each simulation will do 1000000 requests. And the probability of each object could be requested follows the Zipf's law distribution, as described in Section 3.

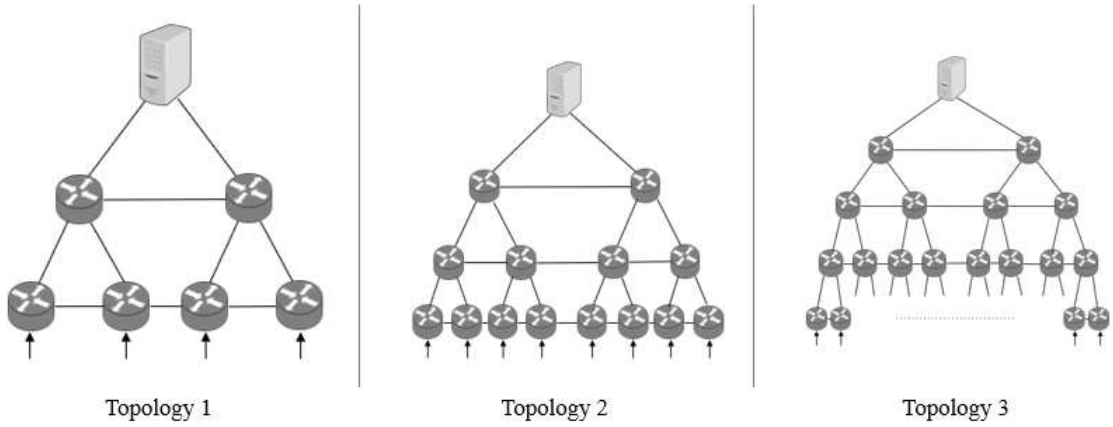

Figure 4. Three topologies used in the simulation.

5.1 Average Hops of All Requests

First, we set the capacity of each router to be 150 (15% of total items in origin server). And the results are as shown in Table 1.

Table 1

Average Hops (C = 150, Request Number = 1000000)			
	Topology 1	Topology 2	Topology 3
Basic	1.46	2.12	2.74
Big Cache	1.17	1.70	2.17
Improved Big Cache	1.10	1.51	1.90

From Table 1 we can see that as the layers number grow from 3 in topology 1 to 5 in topology 3, the average hops number increases in both three strategies. When we dive deep into the numbers, however, we can get more information about the changes. First, by comparing the basic strategy and 'Big Cache', it can be found that the second one is 19.8%, 19.8% and 20.8% faster than the first one in topology 1, topology 2 and topology 3 respectively. And the comparison between Improved Big Cache and Big Cache shows that Improved Big Cache is 5.9%, 11.2% and 12.4% faster than Big Cache in topology 1, topology 2 and topology 3 respectively. And trend implied by the data in Table 1 is that the more complicate the topology of the network is, the more improvement the Improved Big Cache can achieve.

Table 2

Average Hops (C = 100, Request Number = 1000000)			
	Topology 1	Topology 2	Topology 3
Basic	1.57	2.29	2.99
Big Cache	1.29	1.82	2.39
Improved Big Cache	1.21	1.66	2.06

Then, we set the capacity of each router to be 100 (10% of total items in origin server). And the results are as shown in Table 2.

By comparing the data in Table 1 and Table 2, it can be found that the average hops increased in Table2. The reason caused this change is the capacity of the nodes decreased to 10 from 15. According to the detailed algorithms in Section 4, some comparatively less popular objects will have to be moved the upstream layers due to the decrease of the capacity, and the consequence is the increase of the average hops. Hence, this change is reasonable and expected.

If we do the same analysis of the data in Table 2 as we did in Table 1, the same trend can also be found. And Improved Big Cache achieved the same performance when the capacity of the nodes reduced over 30%. In other words, the change of the capacity of the nodes do not have much negative effects on the performance of Improved Big Cache.

5.2 Average Hops of Each Object

Section 5.1 shows the overall performance improvement and steadiness of the Improved Big Cache strategy. And we can also check the simulation results from the view of individual objects. Figures 5-7 shows the result of average hops of each object in three different topologies and two different node capacities. But one thing needs to mention is that all the figures blow shows only the average hops of the first 150 objects, and the average hop numbers of the rest objects are usually the hop number between the edge server and origin server. if we go back to the figure of Zipf's law, we can find that all the object with name greater than 130 has the appearance of probability less than 0.1%. In other words, those objects are rarely requested. Hence, in all the schemes, those items are less popular and highly likely going to be replaced or pushed upstreaming to higher level.

By comparing those two sets of figures in Figure 5-7, there are several properties of those schemes can be easily found.First, from the Basic scheme to Big Cache scheme and then Improved Big Cache scheme, the number of objects with less average hops increased massively. Take topology 3 and capacity is 15% as an example, it can be found that in Basic strategy, the objects with names greater than 35 starts to have average hops of 4, and in Big Cache strategy the objects have average hops of 4 have name greater than 50. The situation in Improved Big Cache is clearer, only the objects with name greater than 110 have average hops of 4. And the reason behind this is the mechanism in Big Cache scheme and Improved Big Cache that pushes the popular objects downstream towards or into the edge servers. As those most popular objects mostly located in lower layer of the network, the overall average hops are hence going down, even though the unpopular objects have high average hops number as stated in the first point.

Second, even though the topologies and node capacities changes, the performance of Improved Big Cache steadily stays as the best in all the three schemes. And it has the least average hops increment among all three strategies when the system model getting more layers and the node capacity decreased.

Hence, from the three sets of figures, it can be found that Big Cache performs better than Basic scheme, and Improved Big Cache has the best performance in all aspects. To sum up, Improved Big Cache has the lowest average hops and largest number of popular objects with low average hops. Also, it performs steady in different topologies.

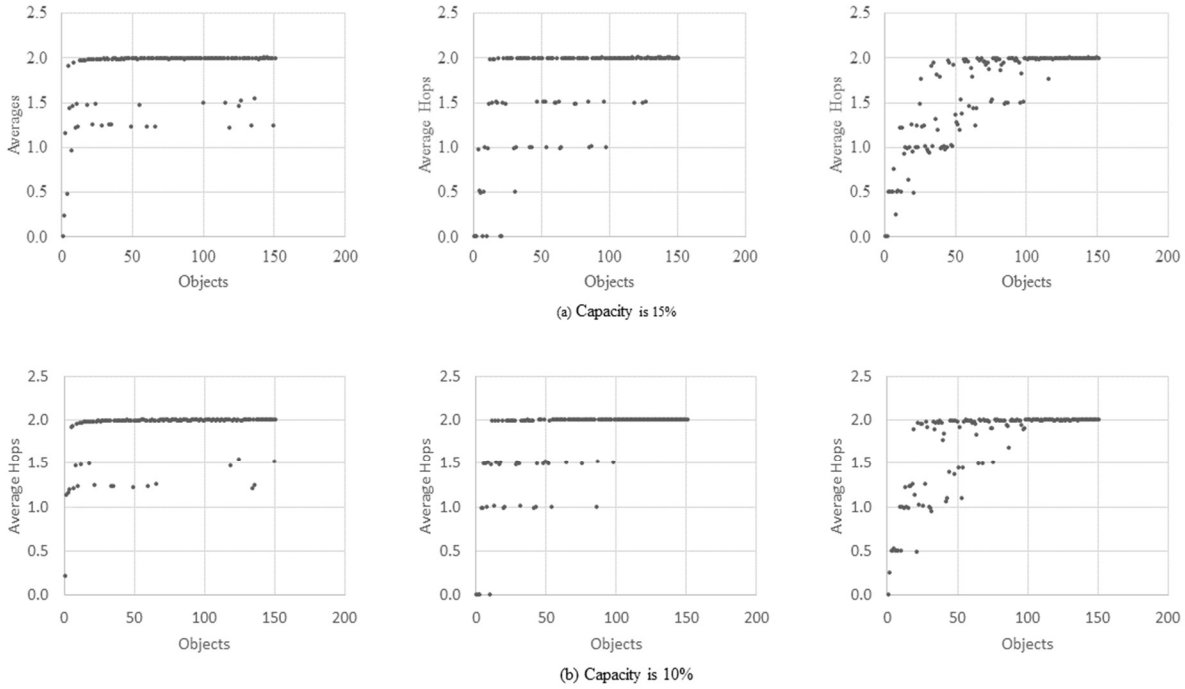

Figure 5. Simulation results in topology 1. The strategies from left to right are Basic, Big Cache and Improved Big Cache

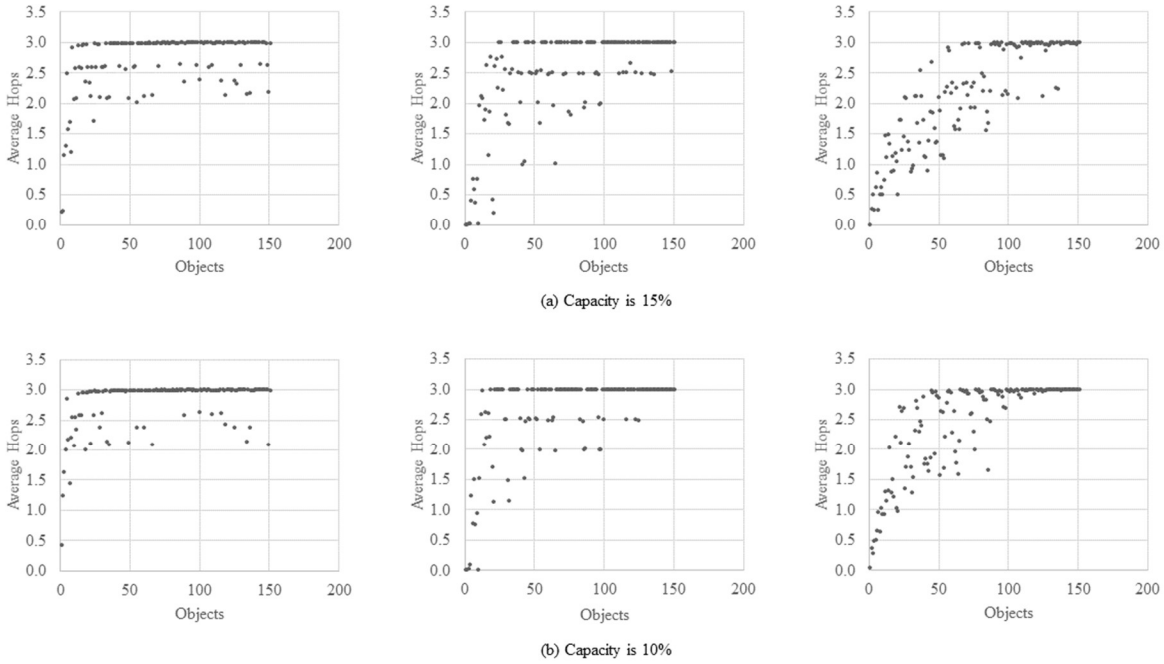

Figure 6. Simulation results in topology 2. The strategies from left to right are Basic, Big Cache and Improved Big Cache

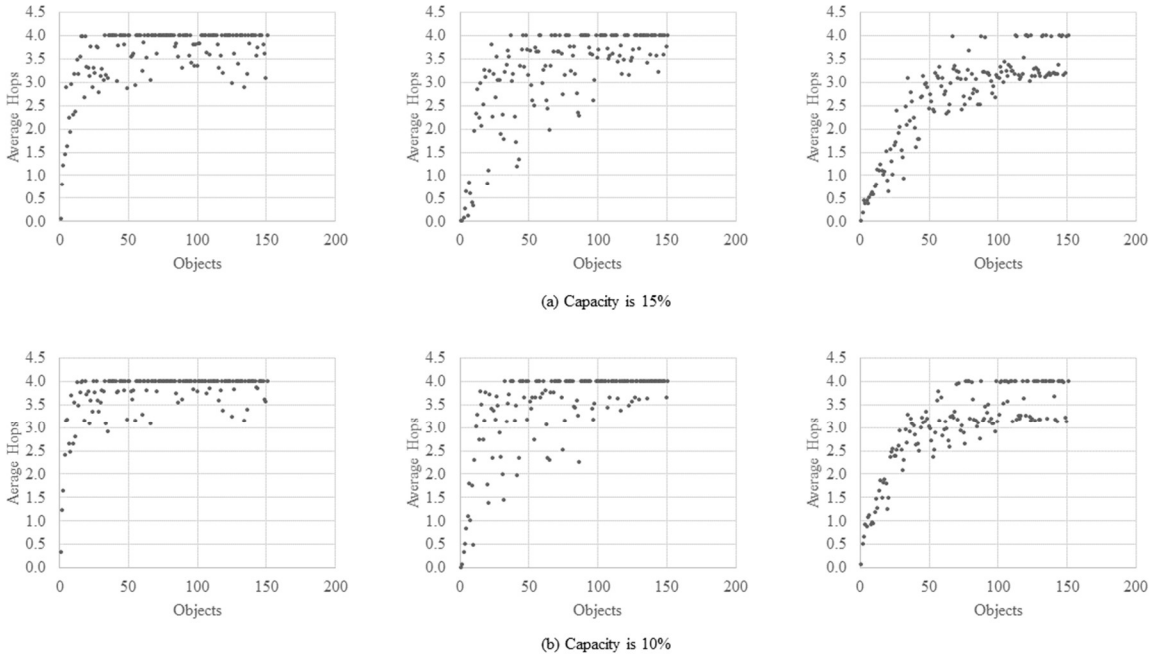

Figure 7. Simulation results in topology 3. The strategies from left to right are Basic, Big Cache and Improved Big Cache

6 Conclusion and Future Work

In this report, each node (except the origin server) combined with its left neighbor and right neighbor, if they exist, to form a bigger cache to improve the hit ratio. Also, a new cache management scheme is proposed. Unlike the usual replacement/eviction strategy, within this scheme the popular objects can pop downstream, i.e., closer to or going into the edge servers. In this way, on each path from edge server to origin server, the popular items are always taking less hops to find them. And consequence of this fact is that, even though for some unpopular objects the hops number are high, the average hops number of the whole requests are low. And this means the network traffic is reduced by applying this scheme.

As for the future work, one interesting thing we can think about is the relationship between the objects from the same file or publisher. Take a video as an example, a video is split into a lot of chunks, when the named ‘N’ chunk is requested, then the ‘N+1’ chunk is highly likely to be requested in the next step, thus it will be very helpful if we can locate or prepare the ‘N+1’ chunk when the ‘N’ chunk is requested.

Reference

[1] Cisco visual networking index: forecast and methodology: 2017-2022, May 2017

[2] G. Zhang, Y. Li, T. Tao. Caching in information centric networking: A survey. 2013.. Computer Networks 57 (2013) 3128-3141.

[3] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, B. Ohlman. A Survey of Information-Centric Networking. 2014. IEEE Communication Magazine July 2012.

[4] B. A. Alcardo, X. Tan. Caching and Data Routing In Information Centric Networking (ICN): The Future Internet Perspective. 2014. International Journal of Advanced Research in Computer Science and Software Engineering. Volume 4, Issue 11, November 2014.

[5] A. V. Vasilakos, Z. Li, G. Simon, W. You. 2015. Information centric network: Research challenges and opportunities. 2015. Journal of Network and Computer Application 52 (2015) 1-10.

[6] I. U. Din, M. K. Khan, O. Ghazali. Caching in Information-Centric Networking: Strategies, Challenges, and Future Research Directions. 2018. IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 20, NO. 2, SECONG QUARTER 2018.

[7] V. Jacobson, D. K. Semetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard. Networking Named Content. 2009. CoNEXT'09, December 1-4, 2009.

[8] J. Pfender, A. valera, W. K. G. Seah. Performance Comparison of Caching Strategies for Information-Centric IoT. 2018. 5th ACM Conference on Information-Centric Networking (ICN'18), September 21-23, 2018.

[9] S. Ioannidis, E. Yeh. Jointly Optimal Routing and Caching for Arbitrary Network Topologies. 2018. IEEE Journal on Selected Areas in Communications. Volume 36, Issue 6, June 2018.

[10] E. Ramadam, A. Narayanan, Z. Zhang. BIG Cache Abstraction for Cache Networks. 2017. 2017 IEEE 37th International Conference on Distributed Computing Systems.

[11] N. Laoutaris, S. Syntila, I. Stavrakakis. Meta algorithms for hierarchical web caches. 2004. In Proceedings of the 2004 IEEE International Performance. Computing and Communications Conference, 2004, pp. 445–452.

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

[13] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, N. Nishinaga. CATT: potential based routing with content caching for ICN. 2012. ICN (2012).

[14] I. Psaras, W.K. Chai, G. Pavlou. Probabilistic in-network caching for information-centric networks. 2012. ICN (2012).

[15] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker. Web caching and zipflike distributions: evidence and implications. 1999. IEEE INFOCOM'99 1 (1999) 126–134.

[16] B. Ahlgren. Second NetInf Architecture Description. 4WARD EU FP7 Project, Deliverable D-6.2 v2.0, April 2010, FP7-ICT-2007-1-216041-4WARD/D-6.2.

[17] M. Ain. D2.3-Architecture Definition, Component Descriptions, and Requirements, Deliverable, PSIRP 7th FP EU-funded Project, February 2009.

[18] T. Koponen, M. Chawla, B.G. Chun, A. Ermolinskiy, K.H. Kim, S. Shenker, I. Stoica, A data-oriented (and beyond) network architecture. 2007. ACM SIGCOMM, 2007.

[19] K. Cho, M. Lee, K. Park, T.T. Kwon, Y. Choi, S. Pack. 2012. WAVE: popularity based and collaborative in-network caching for content-oriented networks. IEEE INFOCOM Workshop on NOMEN, 2012.

[20] Y. Li, T. Lin, H. Tang, P. Sun. A chunk caching location and searching scheme in content-centric networking. 2012. ICC (2012).

[21] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, H. Karl. Network of information (netinf) – an information-centric networking architecture. 2010. Comput Commun 2010a; 36(7).

[22] T.M. Wong, J. Wilkes, My cache or yours? Making storage more exclusive. 2002. Usenix Association Proceedings of the General Track, 2002, pp. 161–175.

[23] D. Lagutin, K. Visala, S. Tarkoma. Publish/subscribe for internet: PSIRP perspective. 2012. IOS Press, 2012.

[24] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker. Less pain, most of the gain: Incrementally deployable ICN. 2013. SIGCOMM, August 12–16, 2013.

[25] S. Borst, V. Gupta, A. Walid. Distributed caching algorithm for content distribution networks. 2010. IEEE INFOCOM, 2010.

[26] J. Dai, Z. Hu, B. Li, J. Liu, B. Li. Collaborative hierarchical caching with dynamic request routing for massive content distribution. 2012. IEEE INFOCOM, 2012.

[27] G. Carofiglio, L. Mekinda, and L. Muscariello. Joint forwarding and caching with latency awareness in information-centric networking. 2016.Computer Networks 110 (2016), 133–153.

[28] R. Chiocchetti, D. Rossi, G. Rossini, G. Carofiglio, and D. Perino. Exploit the known or explore the unknown?: Hamlet-like doubts in icn. 2012.In Proceedings of the second edition of the ICN workshop on Information-centric networking. ACM, 7–12.