An ICN In-Network Caching Policy for Butterfly Network in DCN

Hongseok Jeon[1], Byungjoon Lee[1], Hoyoung Song[1] and Moonsoo Kang[2]
[1] Communications Internet Research Laboratory, Electronics and Telecommunications Research Institute
138 Gajengno, Yuseong-gu, Daejeon - KOREA
[e-mail: jeonhsg@etri.re.kr, bjlee@etri.re.kr, hsong@etri.re.kr]
[2] School of Computer Engineering, Chosun University
Gawangju - KOREA
[e-mail: mskang@chosun.ac.kr]
*Corresponding author: Moonsoo Kang

Received December 17, 2012; revised March 18, 2013; accepted July 5, 2013; published July 30, 2013

Abstract

In-network caching is a key component of information-centric networking (ICN) for reducing content download time, network traffic, and server workload. Data center network (DCN) is an ideal candidate for applying the ICN design principles. In this paper, we have evaluated the effectiveness of caching placement and replacement in DCN with butterfly-topology. We also suggest a new cache placement policy based on the number of routing nodes (i.e., hop counts) through which travels the content. With a probability inversely proportional to the hop counts, the caching placement policy makes each routing node to cache content chunks. Simulation results lead us to conclude (i) cache placement policy is more effective for cache performance than cache replacement, (ii) the suggested cache placement policy has better caching performance for butterfly-type DCNs than the traditional caching placement policies such as ALWASYS and FIX(P), and (iii) high cache hit ratio does not always imply low average hop counts.

Keywords: Information-centric networking, Data center network, In-network caching, On-path caching

This research was funded by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013.

http://dx.doi.org/10.3837/tiis.2013.07.005

1. Introduction

New paradigm of networking architecture called information centric networking (ICN) has been extensively investigated in the context of DONA [1], CCN [2], PSIRP [3], and SAIL [4] projects with a view to overcome the existing Internet limitations due to mass content distribution [5]. Unlike currently used host-to-host addressing, these solutions are based on named data object (NDO) that uniquely represent a piece of content such as a Web page, video clip, document, and so on. In order to get content over ICN, users just request the content itself by using its NDO; they don't have to know where the content is actually located. It is the ICN that automatically recognizes the request and forwards it to real location of the content.

A distinctive ICN feature is that content can be safely relocated as long as its integrity is maintained. Thus, ICN greatly facilitates content replication over physical networks to leverage in-network caching, so that any node storing the content can serve the request, not just the original server. Effectively, in-network caching that stores the copy near to the client helps reducing response time and network traffic among Internet service providers (ISPs).

Speaking of caching, it became a *de facto* standard to install a Web proxy server of the content delivery network (CDN) at the gateway connecting users to the ISP. If the proxy already has the requested content at hand, it immediately responds to the request. Generally, Web proxying is comparable to ICN in-network caching. Proxy servers, however, cannot be regarded as a part of layered network architecture since their maintenance explicitly involves administration overheads.

In contrast, ICN in-network caches, referred to as on-path caches[1], are the integral part of the network – they keep popular contents with no need of additional administration. Furthermore, on-path caching differs from the Web proxying in that each cache has a limited amount of storage, typically RAM-based, where packets are stored. This may cause, however, content fragmentation: content chunks can become scattered over a few neighboring caches. Even in case of a single cache, an individual chunk can be replaced with the chunk from the other content due to the limited amount of memory. In addition, a single chunk can be either stored in a packet or divided into a series of packets, one or some of which can be lost in transmission that further causes chunk fragmentation. Thus, ICN strongly recommends that a single chunk were packed into an individual packet to reduce fragmentation as much as possible.

Caching policy determining what to store and what to replace further contributes to the effectiveness of the ICN on-path caches. The cache hit rate, i.e. how many requests can be served, depends on the policy. Representative caching policies such as LRU and LFU have been studied in the view of network traffic for general network topology. However, the randomness of the requests from many users makes it difficult to determine which is the best solution.

Recently, instead of trying to find the best caching policy for general ICN, researches on caching policy for specific network topology are gaining more interest, since the regularity of the network can greatly help improve caching performance. Particularly, data center networks (DCN) are a major focus of attention because of big data [6]. Accordingly, this paper aims at two goals. First, we have simulated conventional caching policies for DCN with butterfly

[1] The name refers the memories of the routers along the path from the client to the server that are used for caching the content requested by the user.

topology. Simulation results show the feasibility of one or other policy for DCN. Then, we suggest a new caching policy based on hop counts to map content popularity to the regularity of butterfly network that improves the hit rate of the cache. The idea is based on the fact that the routing path in the butterfly network regularly follows tree topology. Thus, the closer the node is to the root, the more users it serves. Accordingly, the closer to the root we put popular content, the better will be the cache hit. The drawback of this approach, however, is that the response time is increasing. Therefore, our hop count based caching is also designed to consider the response time, not just simply putting popular content closer to the root. The effectiveness of the hop count based caching with a view of hit or miss rate and response time was extensively evaluated through repeated simulations.

This paper is organized as follows. In Section 2 we briefly discuss traditional caching policies and their effectiveness for ICN. In Section 3 we present a hop count based caching policy for butterfly network in DCN. Section 4 describes the simulation environments. Section 5 shows the performance evaluation of different caching policies in butterfly networks and their comparison to the suggested method. Finally, Section 6 concludes the paper and outlines future works.

2. Caching Policies

2.1 Cache Placement Policy

Cache placement means a decision of whether to store the newly arrived content chunks in the cache memory. Originally, cache was introduced to support operating systems and databases. For these applications, cache placement policy was not an important design factor in terms of perfomance. However, when it comes to networks [7][8], cache placement policy becomes a critical design factor.

The most straightforward cache placement policy is to place every incoming content chunks into the cache memory. Hereinafter we refer this as ALWAYS policy. Many caching methods use ALWAYS because of its simplicity. However, this approach impedes the distribution of content chunks accross the network because popular content monopolizes the limited available cache.

To reduce monopolization one can cache content chunks with a fixed probability. We refer this as FIX (P). Here, routing nodes determine whether to cache the passing content chunks based on fixed probability P whose value depends on network size and content popularity. In general, the optimum value of P is acquired empirically.

Laoutaris [9] proposed a cache placement policy based on content popularity in hierarchical networks where caches are located at different network levels such as regional, national, and so on. WAVE [10] further generalized the popularity-based cache placement policy regardless of specific network topology. In WAVE, upstream routing nodes explicitly suggest what content chunks must be cached at the next downstream routing nodes. Consequently, as the content becomes more popular, the copies of the corresponding content chunks become closer to users.

Dong [11] has developed a mathematical model for optimized cache placement on the assumption that the routing node knows what content chunks are actually cached at the neighboring nodes. In [11], each routing node makes a decision as to cache in order to minimize the average content retrieval latency subject to limited capacity of individual node cache.

2.2 Cache Replacement Policy

Cache replacement refers to the decision of which cache entry should be removed from memory in order to free space for new incoming content chunks when cache memory is full.

Most cache replacement policies take advantage of the locality of the reference. The least recently used (LRU) and the least frequently used (LFU) policies are well-known examples. LRU takes into account temporal locality which suggests that recently used content chunk is likely to be reused soon. Accordingly, LRU removes content chunks based on how long they had not been accessed. LFU further suggests that the probability of reuse increases with the number of references. Over the years, many researches have been made to extend classical policies [12]. However, there are little works on LRU and LFU in the context of on-path caching.

Locality-based replacement policy essentially assumes the skewed popularity of contents, and tries to keep popular content chunks resident in the cache. On the contrary, Rossi [13] proposed a cache replacement policy, called BIAS, to cache diverse content. When the cache becomes full, BIAS randomly selects two chunks and removes the more popular one.

Compared to general caching, on-path caching is different in that cached data can belong to different original storages. Therefore, when cached content is not available, the distance required to fetch the corresponding data from the original server can be different. The distance travelled on the network immediately affects the cost of network traffic. For Web environment, Peter [14] argued that a cache replacement algorithm should minimize the cost of missing cache and proposed LNC-R-W3 delay-conscious cache replacement algorithm. Recently, Wang [15] also proposed a cache replacement policy which takes the distance into account for in-network caching.

While the aforementioned caching replacement policies use only historical information, Famaey [16] showed that theoretically a prediction-based caching policy may be of great advantage, provided good popularity prediction is possible.

3. Caching Policy for Butterfly DCN

DCN is considered as a good candidate to apply ICN concept for its unique features such as server consolidation, workload distribution, and traffic offloading [6]. In server consolidation, a few servers are grouped into a single virtual server to provide a specific service. If a particular server becomes overloaded for some reason, its content or services are replicated to its neighbors to reduce the workload. At the same time, traffic congestion among the servers can also be a burden for the DCN. To reduce the east-west traffic in the DCN, data caching for offloading the traffic is emphasized. Thus, in-network caching can be used on a systematic base for self-contained DCN to cover load balancing and the cache proxying.

3.1 Butterfly Network

Thousands servers that form a DCN are usually organized in traditional 2N tree topology such as VL2, PortLand, DCell, and BCube [17]. Even if the resulting topology looks like a mesh rather than a tree (which can be the case to increase redundant connections,) the resulting mesh still can be considered as multiple overlapping trees. For an example, a fat tree, popularly used in a DCN, is constructed by more than two trees allowing multi-path routing on multiple connections between parent and child nodes.

Starting from the observation, we can approximate a butterfly network with a combination of binary tree networks. Also, we believe a single binary tree is enough to analyze and evaluate the effect of the existing caching polices, because the number of incoming connections onto a node only affects the content request distribution. For example, three or four incoming connections to a node, compared to two connections, will have different distribution of content requests, but no different topological feature. Moreover, a caching policy on each node so separately works with each other that the resulting performance will be driven depending on the distributions, not on the number of connections.

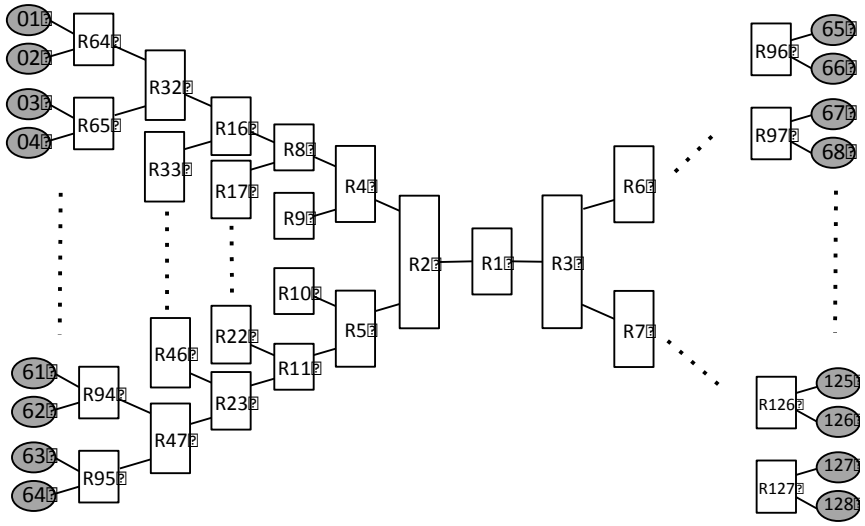


Fig. 1. A buttefly network approximated with a perfect binary tree

3.2 Cache Placement Policy based on Hop Counting

In this paper we suggest a simple but efficient cache placement policy that takes the advantage of the regularity of the binary tree whose subgraph[2] is always another smaller binary tree. In other words, any node in the tree will be a root node of a binary tree made of all its child nodes. Because regularity at each location may inherently reflect the level of aggregation of content requests, we are using location information, that is the number of hops from the server to the router toward the client as a factor to stochastically select and cache more popular content chunks.

Fig. 1 shows an example of how content popularity can be mapped to the regularity in a binary tree. We assume that the root node (R1) is a content server and leaf nodes are clients requesting the content stored on the server. Then, the routing protocol makes the internal nodes from a leaf node to the root node intermediate routing nodes. From the properties of the binary tree, the node at the level n has (tree depth − n) hops and $2^{\text{tree depth}-n}$ leaf nodes. For example, in the case of binary tree with depth 7 as shown in **Fig. 1**, internal nodes with the hop counts of 1 and 2 have 32 and 16 leaf nodes, respectively. Therefore, a node with a smaller hop count can cover a larger number of users. Accordingly, the more users request the content, the closer to the server should be the node where it is placed. This aggregation makes it possible to

[2] We use the term of 'subgraph' not referring to a part of graph but a smaller tree in the paper.

design a hop count based cache placement (HCCP) policy, by which each routing node determines whether to cache the incoming content chunk with a probability of 1/(hop count). Simply stated, as a router gets away from the server and the hop count increases, the caching probability goes down. Alternatively, caching probability increases. To realize HCCP[3] policy, content chunks should include a reserved field for counting the number of hops from the server.

Let's see how HCCP works by way of example. When content request arrives at the server, the server forwards the corresponding content chunks to the first router whose hop count is 1. Thus, the probability to cache the chunks on the first router is 1.0, and the content is always cached. If the cache is already full, one item will be removed using LRU or LFU algorithms. Thus, at the first router, the HCCP works like ALWAYS. If content chunks are further forwarded to the second router, the probability to cache them is halved to 0.5 because the corresponding hop count is 2. Whether the chunks are cached or not will be determined by fifty-fifty algorithm that is the same as FIX (0.5). As the chunks are forwarded to the next routers, the procedure repeats. It differs from ALWAYS and FIX (P) in that HCCP caching probability decreases as the chunks are as closely forwarded to the user.

We can characterize the HCCP caching behavior depending on the location of each router as following. A node closer to the root will serve more users. More users will produce more varied content requests, whereby the popularity of a specific content cannot be easily biased. As the popularity of the specific content among the users becomes more random, lengthily retaining an old chunk on the cache will not be beneficial. It is natural to replace the chunks on the cache frequently with higher caching placement probability of a new chunk as HCCP suggests.

On the contrary, a node farther away from the root has lower caching probability and serves fewer users. Even though few users can trigger diversified content requests, the common popularity of a specific content can be easily observed and it longer lasts than that of more users. In this case, it makes sense to retain old chunks further. It is natural to replace the chunks on the cache rarely with lower caching placement probability of a new chunk as HCCP suggests.

4. Theoretical Analysis

Due to the statistical dynamics, it is very difficult to demonstrate HCCP advantages theoretically. But we can show, however, why HCCP has better cache hit rate than ALWAYS. We consider an *n*-depth perfect binary tree connected to a server as shown in **Fig. 2** that is an example of the full binary with 8 users.

[3] The HCCP can be easily extended to the arbitrary m-ary tree because the topology cannot restrict the hop counting.

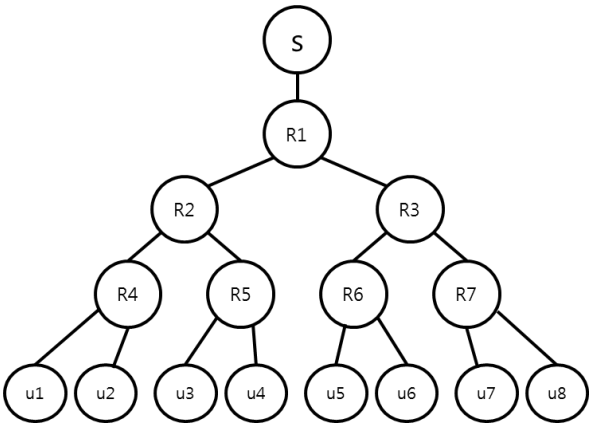


Fig. 2. A full binary tree with 8 users

Let's assume the server has N content, labeled as C_s (S = {1, 2, 3, ..., N}) and each user U_i is requesting the set of content, denoted as C_i (=$\{C^i_1, C^i_2, \ldots C^i_m\}$), where $C_i \subset C_s$. The size of each set may be diffrent and the element C^i_j of C_i is randomly selected from C_s. Probability of requesting C^i_j follows zipf distribution. Let's assume now that each router has its own sized cache. For simplicity, we assume that all routers have the same cache size M. In our analysis, we consider only a situation of saturation where the caches become full after some period of time which is long enough. We also assume HCCP and ALWAYS use LRU as the cache replacement policy.

Theorem 1) the cache hit rate of a cache with the cachig probability p is bigger than that of the cache with the caching probability q if $p < q$ on a given content request distribuiton.

Proof) If the caching probabiliy is p then more than $1/p$ content requests are cached. If the cache has the caching probability p, this will form a set of cached content C_{Ri} (= $\{C^{Ri}_1, C^{Ri}_2, \ldots C^{Ri}_m\}$). If the cache has caching probability q, this will form another set of cached content C_{Ri}^* (= $\{C^{Ri}{}_1^*, C^{Ri}{}_2^*, \ldots C^{Ri}{}_m^*\}$). It is obvious that the number of requests for C^{Ri}_j to be cached is a geometric random variable representing the number of trials until the first success and its expected value is $1/p$. Thus the expected trials to form C_{Ri} and C_{Ri}^* is m/p and m/q respectively. Then, for a given content request distribution,

$$\sum_{j=1}^{m} \Pr\left(C_j^{R_i}\right) \geq \sum_{j=1}^{m} \Pr\left(C_j^{R_i{}^*}\right),$$

where $\Pr(C^{Ri}_j)$ is the probability that the content C^{Ri}_j is requested under the given content distribution. From the above inequality, the cache hit rate of the cache with cachig probability p is bigger than that of the cache with caching probability q if $p < q$. ■

Intuitively, Theorem 1 means the following. If caching probability is 1, then all content requests are immediately cached, which means unpopular content can replace popular content. However, if caching probability gets lower, unpopular content is less likely to replace popular content. For example, if caching probability is 1/4, more than 4 requests may let content cached while less than 4 requests may not let the content cached.

At last, we can conclude that the HCCP cache hit rate is higher than that of ALWAYS from the following inequality,

$$\sum_{l=1}^{n}\sum_{i=2^{l-1}}^{2^{l}-1} CHR^{HCCP}(R_i) \geq \sum_{l=1}^{n}\sum_{i=2^{l-1}}^{2^{l}-1} CHR^{ALWAYS}(R_i)$$

,

where $CHR^{HCCP}(R_i)$ and $CHR^{ALWAYS}(R_i)$ are the cach hit rate of the router R_i. The above inequality is satisfied by Theorem 1 since HCCP caching probability is $1/2^{[\log_2 i]}$ which is always lower than that of ALWAYS.

5. Simulation Results

For simulation, we used NS2 (version 2.35) and a perfect binary tree for approximating the DCN's butterfly network with 127 internal nodes and 128 leaf nodes as shown in **Fig 1**. Internal nodes play the role of routing nodes, and leaf nodes function as both content providers and consumers. Thirty thousand content files were evenly distributed among content servers. For better traceability of the simulation, we assume each content file has a size of 10Mbytes and is divided into 1Kbyte chunks, whereby the degree of chunk distribution for content over caches can be easily observed and analyzed. We also assume the size of chunk should be smaller than the default MTU of the Ethernet interface[4] to avoid a single chunk to be transmitted in a few of packets, which may cause reassembling problems due to packet losses during transmission. The cache size of each routing node is set to 1Mbyte emphasizing the effect of the limited cache size.

To model a skewed popularity of content, we used a Zipf distribution [18] with various values of Zipf rank exponent α. All content is ranked with Zipf distribution. According to survey [13], small values of α (between 0.7 and 1) indicate a lightly loaded Web server and α = 1.5 corresponds to a busy Web server. Large values of α (between 2 and 2.5) mean that the given content gets higher popularity that can be easily observed in YouTube [19]. Nowadays, such extremely skewed popularity temporally happens due to social network effect. Social networks enable content to become highly popular for very short time and some popular sites can bring about great slashdot effect and flash crowd in a very short time [20]. Our simulation runs until thousandth content request occurs. Table 1 summarizes simulation parameters.

Table 1. Simulation parameters

Parameter	Value
Chunk size	1 KB
Cache storage size	1 MB
Content size	10 MB
Content items	30,000
Number of routing nodes	127
Number of content servers / users	128
Zipf rank exponent (α)	0.7, 1, 1.5, 2, 2.5

[4] We also assume a DCN may have a higher or equal speed links to the Ethernet among its nodes. However the link speed is not a matter, it is more important whether a single chunk can be packed into a single packet to avoid chunk fragmentation.

We evaluated the performance of in-network caching policies in terms of average hop counts and cache hit ratio. The cache hit ratio estimates the average probability to find content chunks on the routing nodes before getting down to the content servers. This metric indicates the efficiency and load balancing effect of the in-network caching policy in DCN. The average hop count estimates the average number of hops that content requests from users travel on the network to obtain the requested content chunks. This metric can be used to quantify the expected download time minimized by using the in-network caching.

Fig. 3 plots the cache hit ratio as a function of the Zipf parameter for different cache placement policies: HCCP, FIX (P), and ALWAYS. In this simulation, P in FIX (P) is set to a probability resulting in the best performance[5]. As can be seen, the cache hit ratio is nearly two-times higher in both HCCP and FIX (P) than in ALWAYS (i.e., HCCP and FIX show 6.7 % and 6.3% higher cache hit ratios than ALWAYS at the 2.5 Zipf parameter, respectively). This means that in-network caching using HCCP and FIX (P) reduces the content server workload more than ALWAYS. This result is not surprising because ALWAYS policy caches all content chunks aggressively. The cache hit ratio increases as the Zipf parameter increases.

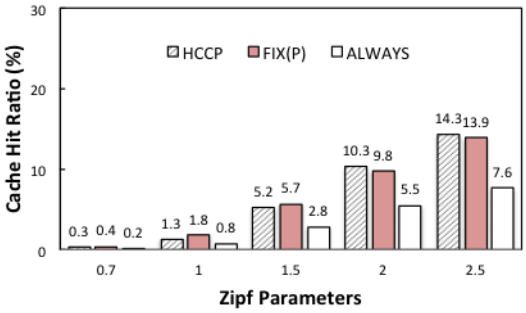


Fig. 3. Cache hit ratio for different cache placement policies

Fig. 4 shows drops in average hop counts using HCCP and FIX (P), compared to ALWAYS. As expected, HCCP and FIX (P) have the benefit of the less average hop counts, and this benefit increases as the Zipf parameter increases. However, they do not show a noticeable difference from ALWAYS in comparison with the cache hit ratio shown in **Fig. 2**. (i.e., neither HCCP nor FIX (P) can decrease even 1 hop in the average hop count). We analyze the cause of this in **Fig. 5**.

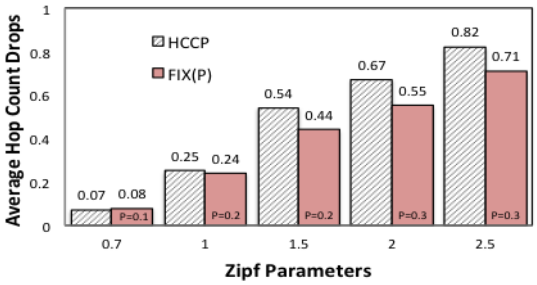


Fig. 4. Drops in average hop counts of HCCP and FIX (P) compared to ALWAYS

[5] We conducted multiple simulations for FIX (P) as we increased the P by 0.1 and we have found that FIX (0.1) shows best performance in $\alpha = 0.7$, FIX (0.2) performs the best at $\alpha = 1$ and 1.5, and FIX (0.3) causes the best performance in $\alpha = 2$ and 2.5.

Fig. 5 shows averages cache hit ratios in routing nodes using HCCP, FIX (0.3), and ALWAYS when α is set to 2.5. The x-axis plots the index number of Ri of each router in **Fig. 1**. As shown in **Fig. 5**, HCCP and FIX (0.3) achieve a compelling cache hit ratio in the core and intermediate routing nodes. HCCP show a slightly better performance than FIX (0.3) for these routing nodes. At routing nodes 11 through 20, HCCP and FIX (0.3) show 11.7% and 10.2% higher cache hit ratios than ALWAYS. However, the cache hit ratio of both HCCP and FIX (0.3) decreases in the edge routing nodes, and HCCP exhibits a poorer performance than FIX (0.3) and ALWAYS in some edge routing nodes. We think that because HCCP and FIX (P) show intensively higher cache hit ratios than ALWAYS in the core and intermediate routing nodes, they cannot provide a remarkable achievement in the average hop counts compared to ALWAYS. In **Fig. 3** through **5**, LRU is used as a common cache replacement policy.

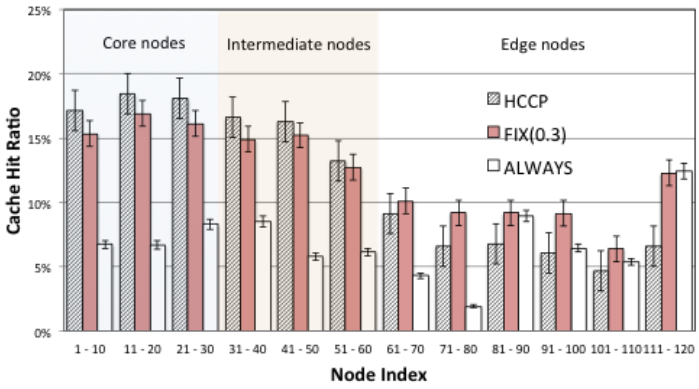
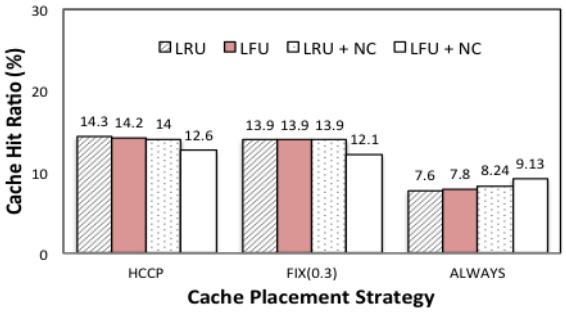


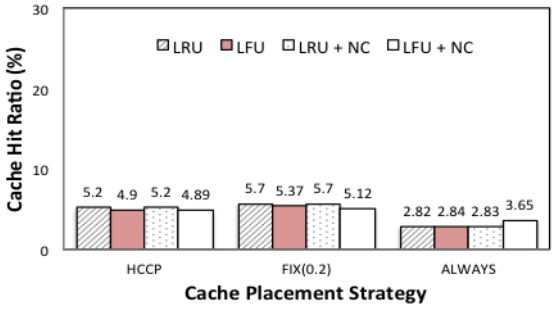
Fig. 5. Cache hit ratio of groups of adjacent routing nodes in α = 2.5

In **Fig. 6**, we explore the impact of four different cache replacement strategies (LRU, LFU, LRU + network cost (NC)[6], and LFU + NC) depending on different value of α. From **Fig. 6**, one can see that LRU and LFU show no noticeable performance difference. LFU generally outperforms LRU for static environments where popularity of data is consistently unchanged over the time [12]. However, in our simulation results, the performance of LFU is quite similar to that of LRU. We are reasoning any advantageous feature of LRU and LFU cannot be dominantly exists since the node at a higher level in the tree may receive more requests from more users and the dominant features like temporal reference or reference frequency are mixed and amortized. We believe such indifference between LRU and LFU would be presented in current Internet because the locality of reference becomes weakened as Web 2.0 comes a multitude of short video clips and social networks are generating dynamic content popularity. In **Fig. 6**, we can see LRU+NC also does not show appreciable difference between LRU and LFU. LRU+NC has just a little impact on HCCP and FIX (P). We believe this is natural because the network cost in LRU+NC is just an adding factor to the recency and thus does not have a big impact on LRU. However, LRU + NC has an impact on the performance compared to others. LFU+NC has a slightly negative impact on both HCCP and FIX (P) and a positive impact on ALWAYS. This is because the network cost in LFU+NC is a multiplying factor to the frequency and this results in somewhat big impact on LFU.

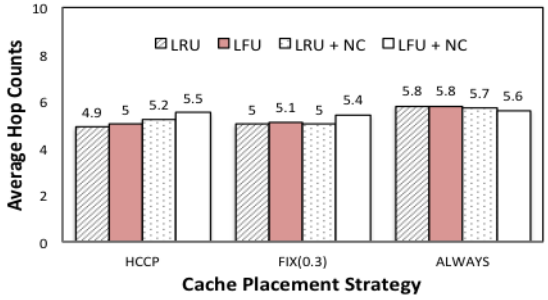
[6] When caching replacement policy determines a target in the cache, a time cost to get the target from the source server again should be accounted together unlikely to only considering how much frequently or recently used in LRU and LFU [15].



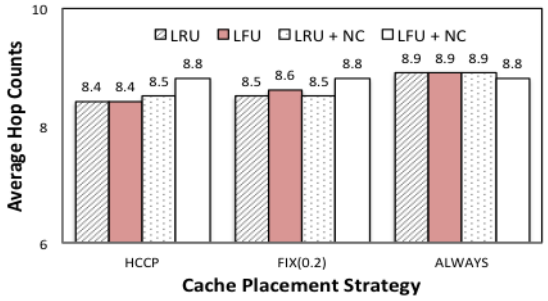
(a) α = 2.5



(b) α = 1.5



(c) α = 2.5



(d) α = 1.5

Fig. 6. Average hop count and cache hit ratio for different cache replacement policies

5. Conclusion

This paper focuses on the impact of caching placement and replacement policies on the performance of in-network caching for DCN butterfly networks. We analyzed the HCCP behavior and theoretically compared it with ALWAYS. This explains why HCCP has better cache hit rates than ALWAYS. We also performed various simulations to compare three cache placement policies and four cache replacement policies. The results show that it is important to choose the right cache placement policy for the in-network caching performance. In particular, the proposed HCCP shows better performance than other cache placement policies at larger Zipf parameters without significantly longer delay than the others. In addition, cache replacement policies do not show noticeable performance difference. Remarkably, the results show that the high cache hit ratio does not always imply low average hop counts. We have observed FIX (P) and ALWAYS as well as HCCP are causing the phenomenon. Hence, despite a high cache hit ratio, content consumers cannot receive a high level of QoE (i.e., reduced content download time). To lessen the average hop count, some interworking between routing nodes might be required. In the future, we plan to further investigate such interworking methods to considerably decrease the average hop counts. At the same time, we plan to perform evaluations of HCCP on more general network topology with multiple routing paths, not limited on a specific topology.

References

[1] T. Koponen, et al., "A Data-Oriented and Beyond Network Architecture," in *Proc. of Conference SIGCOMM '07 ACM SIGCOMM 2007 Conference*, pp. 181-192, August, 2007.
Article (CrossRef Link)

[2] V. Jacobson, et al., "Networking Named Content," in *Proc. of the 5th International Conference on Emergin Networking Experiments and Technologies*, pp. 1-12, December, 2009.
Article (CrossRef Link)

[3] A. Zahemszky, Csaszar Andras, P. Nikander and C.E. Rothenberg, "Exploring the Pub/Sub Routing & Forwarding Space," in *Proc. of IEEE ICC Workshops 2009*, pp. 1-6, June 14-18, 2009.
Article (CrossRef Link)

[4] Scalable and Adaptive Internet Solutions, http://www.sail-project.eu/

[5] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A Survey of Information-Centric Networking," *IEEE Communications Magazine*, vol.50, no.7, pp.26-36, July 2012.
Article (CrossRef Link)

[6] B. J. Ko, et al., "An Information-Centric Architecture for Data Center Networks," in *Proc. of 2nd edition of the ICN Workshop on Information-Centric Networking*, pp. 79-84, August 13-17, 2012.
Article (CrossRef Link)

[7] S. Bhattacharjee, K. L. Calvert and E. W. Zegura, "Self-Organizing Wide- Area Network Caches," in *Proc. of IEEE INFOCOM '98*, vol. 2, pp. 600-608, March 29 – April 2, 1998.
Article (CrossRef Link)

[8] X. Tang and S. T. Chanson, "Coordinated En-Route Web Caching, Journal." *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 595-607, June, 2002.
Article (CrossRef Link)

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

[10] K. Cho, et al, "WAVE: Popularity-Based and Collaborative In-Network Caching for Content-Oriented Networks," in *Proc. of 2012 IEEE Conference on Computer Communications*

Workshop (INFOCOM Workshop), pp. 316-321, March 25-30, 2012.
Article (CrossRef Link)
[11] L. Dong, Dan Zhang, Y. Zhang and D. Raychaudhuri, "Optimal Caching with Content Broadcast in Cache-and- Forward Networks," in *Proc. of 2011 IEEE International Conference on Communications (ICC)*, PP 1-5, June 5-9, 2011.
Article (CrossRef Link)
[12] S. Podlipnig and L. Boszormenyi, "A Survey of Web Cache Replacement Strategies," *Journal ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 374-398, December, 2003.
Article (CrossRef Link)
[13] D. Rossi and G. Rossini, "Caching Performance of Content Centric Networks under Multi-Path Routing (and More)," *Technical report of Telecom ParisTech*, 2011.
Article (CrossRef Link)
[14] P. Scheuermann, J. Shim, and R. Vingralek, "A case for delay- conscious caching of Web documents," *Journal Computer Networks and ISND Systems*, vol. 29, no 8-13, pp. 997-1005, September, 1997.
Article (CrossRef Link)
[15] S. Wang, et al., "Could In-Network Caching Benefit Information-Centric Networking?," in *Proc. of the 7th Asian Internet Engineering Conference*, pp. 112-115, 2011.
Article (CrossRef Link)
[16] J. Famaey, T. Wauters, and F. D. TURCK, "On the Merits of Popularity Prediction in Multimedia Content Caching," in *Proc. of 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 17-24, May 23-27, 2011.
Article (CrossRef Link)
[17] B. Heller, et al., "ElasticTree: Saving Energy in Data Center Networks," in *Proc. of the 7th USENIX conference on Networked systems design and implementation (NSDI 2010)*, pp.17-17, 2010.
Article (CrossRef Link)
[18] L. A. Adamic and B. A. Huberman, "Zipf's law and the Internet," *Journal Glottometris*, vol. 3, no.1, pp. 143-150, 2002.
Article (CrossRef Link)
[19] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn and S. Moon, "I tube, you tube, everybody tubes: analyzing the worlds largest user generated content video system," in *Proc. of the 7th ACM SIGCOMM Conference on Internet Measurement*, pp. 1-14, 2007.
Article (CrossRef Link)
[20] C. Canali, M. Colajanni and R. Lancellotti, "Characteristics and Evolution of Content Popularity and User Relations in Social Networks," in *Proc. of IEEE Symposium on Computers and Communications (ISCC)*, pp. 750-756, June 22-25, 2010.
Article (CrossRef Link)

Hongseok Jeon received the B.S degree in Industrial Engineering from Sung Kyun Kwan University, Seoul, Korea, in 2002. He received the M.S. degree in Engineering from Information and Communications University (ICU), Daejon, Korea, in 2004. He joined Electronics and Telecommunications Research Institute (ETRI) in 2004. Currently he is a Senior Engineer in the Smart Node Platform Research Section, Smart Network Research Department, Communications Internet Research Laboratory, ETRI. He is interested in the Information-Centric Networking (ICN) and Smart Internet.

Byungjoon Lee received his B.S. and M.S. degrees in Computer Engineering from Seoul National University in 1996 and 1998, and received Ph.D. degree in Computer Engineering from Chungnam National University in 2011. He joined Electronics and Telecommunications Research Institute (ETRI) in 2001, where he has served as a software engineer. Currently he is a Senior Engineer in the Software Defined Network Research Section, Future Internet Research Department, Communications Internet Research Laboratory, ETRI. He is interested in the Future Internet Technologies, including Information-Centric Networking (ICN), and Software-Defined Networking (SDN).

Hoyoung Song is currently working as the director of Research Planning Team for Communications & Internet in ETRI, Rep. of Korea. He received PhD. degrees in information & communication engineering from Chungbuk National University in Korea. He has been with ETRI since 1983. His recent research interests include Information Centric Networking, Cloud networking, Software Defined Networking and future internet technologies.

Moonsoo Kang received the B.S degree in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea, in 1998. He received the M.S. and Ph.D. degrees in Engineering from Information and Communications University (ICU), Daejon, Korea, in 2000 and in 2007 respectively, which is currently a part of KAIST. Since 2007, He is an associate professor with the School of Computer Engineering, Chosun University in Gwangju, Korea. His research interests are various network protocols on ad hoc networks such as sensor, mesh and vehicular networks.