WAVE: Popularity-based and Collaborative In-network Caching for Content-Oriented Networks

Kideok Cho, Munyoung Lee, Kunwoo Park, Ted Taekyoung Kwon, Yanghee Choi
School of Computer Science and Engineering
Seoul National University, Seoul, Korea
Email: {kdcho, mylee, kwpark, tk, yhchoi}@mmlab.snu.ac.kr

Sangheon Pack
School of Electrical Engineering
Korea University, Seoul, Korea
Email: shpack@korea.ac.kr

***Abstract*—In content-oriented networking, content files are typically cached in network nodes, and hence how to cache content files is crucial for the efficient content delivery and cache storage utilization. In this paper, we propose a content caching scheme, WAVE, in which the number of chunks to be cached is adjusted based on the popularity of the content. In WAVE, an upstream node recommends the number of chunks to be cached at its downstream node, which is exponentially increased as the request count increases. Simulation results reveal that the average hop count of content delivery of WAVE is lower than other schemes, and the inter-ISP traffic volume of WAVE is the second lowest (CDN is the lowest). Also, WAVE achieves higher cache hit ratio and fewer frequent cache replacements than other on-demand caching strategies.**

***Index Terms*—Content chunk caching, caching strategy, content-oriented networks**

I. Introduction

For efficient content delivery, end hosts should be able to exploit closer/multiple copies of the requested content. To address such issues, content delivery network (CDN) technologies and application-specific solutions like P2P have became so popular. However, content providers should pay the cost to use CDN services, and CDN may experience sub-optimal performance due to the traffic engineering of Internet service providers (ISPs) [1]. P2P systems incur a lot of inter-ISP traffic and are unstable in terms of content availability and download performance. To overcome these limitations and support content retrievals efficiently, content-oriented networking architectures are proposed where network entities such as routers (or co-located storage servers) cache files [2], [3], [4], [5]. By exploiting in-network storages (and content-based routing in some cases), they provide faster content delivery and offer better content availability than the current Internet.

There are many issues to be solved such as architectural designs [2], [3], [4], [5], content name-based routing [6], [7], security [2], [3], [8], and legacy application support [9], [10]. We focus on in-network caching. There have been studies on how to populate/maintain in-network storages considering content requests with different popularity and huge in-network storages. For example, in-network storages can be partitioned for different application classes to provide service differentiation [11]. Content types (e.g., uploaded, cached, etc.) can be used to decide which content files will be cached first [12]. Depending on the caching strategies, the performance of content-oriented networking can vary significantly.

As on-demand caching becomes prevalent, a larger file may incur longer transfer delay and more processing overhead for each caching operation (store the file into cache, deliver the file from the cache, etc.). To mitigate the overhead, a file can be divided into small sized chunks in content-oriented networks (e.g., [3]). Chunk-based caching has merits over file-based caching. For instance, different chunks of the same file can be delivered from multiple sources. Replacing some chunks instead of a whole file may increase the storage efficiency. In chunk-based delivery, how to distribute chunks of the same file may be crucial. For instance, to support some applications (e.g., video streaming) that require sequential delivery of chunks, the forepart of a file should be delivered to end hosts faster than the following part. Therefore, it is desirable for a caching scheme to consider the inter-chunk distance as well.

There have been many studies to distribute files efficiently in web caches and CDNs [13], [14], [15], [16], [17]. However, they have limitations to be applied in content-oriented networks directly. For instance, [13], [14], [15], [16] assume the specific topologies such as a tree or hierarchical structure. [17] assumes an explicit coordination between caches, which incurs the substantial communication and maintenance overheads. [15] assumes that the content request pattern is known in advance. Since a large number of caches are to be deployed in content-oriented networks, a specific topological assumption, an explicit and tight coordination, and a priori knowledge on request patterns may not be applicable or affordable.

In this paper, we propose a chunk-based caching scheme, WAVE, that aims at efficient content delivery and cache usage while lowering the overhead of cache management. WAVE distributes content chunks towards end hosts considering the content popularity as well as inter-chunk (distance) relation. The main characteristics of WAVE are summarized as follows.

1) **Popularity-based:** WAVE adjusts the number of chunks to be cached considering the content popularity (i.e., access count). As the access count increases, WAVE exponentially increases the number of chunks to be cached and disseminates them more widely.
2) **Simple:** WAVE requires no knowledge of access patterns a priori. WAVE's caching decision requires only

two counters per file. Also, WAVE can operate with any content routing schemes since it uses only the information from where the content request arrives.

3) **Decentralized:** There is no central server in WAVE since caching decisions are made at individual routers independently.
4) **Incrementally deployable:** In WAVE, an upstream router suggests caching to its downstream router by marking a chunk to be cached. If a downstream router does not wish to store the chunk by any reason, it can ignore the suggestion and leave caching to other downstream routers. Thus, WAVE can be deployed without inter-domain cooperation; even in the same domain, WAVE routers can operate with legacy routers.

The remainder of this paper is organized as follows. In Section II, we describe the WAVE scheme. Section III presents the simulation results. Section IV concludes this paper with future work.

II. WAVE: A Content Caching Scheme

A. Assumptions

We assume that content routers (or C-routers, for short) will cache the chunks of files by exploiting in-network storages [2], [3], [4], [5], [6], [7]. If only a subset of the routers have the storage modules, they can form an overlay caching network among themselves[1]. We also assume original servers (who publish the content files) as well as C-routers can recommend its downstream C-routers to cache the chunks.

As a file is divided into small size chunks, an end host requests the file in the unit of a chunk (e.g., Interest packets in CCN [3]). Thus, a file that consists of 100 chunks will be requested by sending out 100 chunk requests from the host. Also, the chunk index can be identified and associated with the requested file at the C-routers. We believe it is not scalable to have a centralized entity that can monitor and control in-network caching, especially across different domains. Thus, in WAVE, each C-router makes caching decisions (e.g., what to cache and what to replace) independently of other C-routers.

However, we need some collaboration among C-routers to avoid inefficient caching situations (e.g., all C-routers cache the same set of chunks). In WAVE, a C-router suggests caching of a chunk to its downstream C-router by marking the chunk when it forwards the chunk. For this, a *cache suggestion* flag bit is needed in the chunk (e.g., in the Data packet header in CCN [3]). If the downstream router does not have a storage or has its own caching policy (e.g., since it belongs to a different domain), it may ignore the caching suggestion and leave caching to other downstream routers.

For content routing, we assume that a chunk request will be routed towards its original server. If the requested content is cached at a C-router along the path (or discovered by content routing), the chunk will be transferred from the C-router, and the chunk request will not be forwarded towards the original server [16], [17].

[1]WAVE can be extended to support a single cache or an uncooperative cache scenario [18], which will be investigated in our future work.

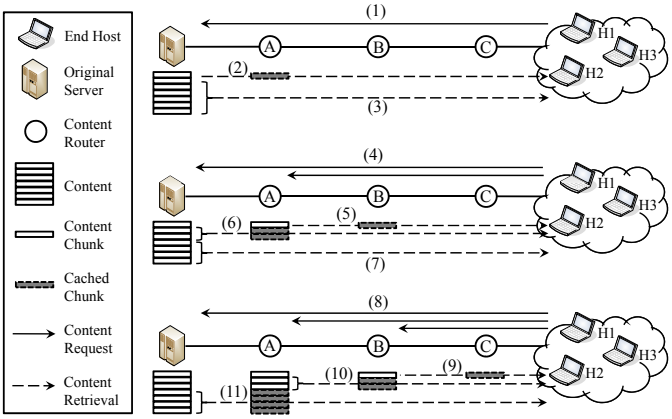


Fig. 1. Illustration of WAVE operations.

B. WAVE Operations

WAVE operations are illustrated in Fig. 1. There are an original server, three end hosts ($H1$, $H2$, $H3$), and three C-routers (A, B, C). Suppose a file consisting of 7 chunks is requested by the three end hosts.

(1) A file request (more precisely, 7 chunk requests) from end host $H1$ will be routed to the original server. (2) The original server will transfer the file to $H1$. Since this is the first retrieval for the content, the original server marks (i.e., sets the *cache suggestion* bit to 1) the first chunk to make it cached, and forwards it to C-router A, from which the file request comes. The first chunk will be cached at C-router A and forwarded to $H1$. Note that the *cache suggestion* bit will be reset to 0 at C-router A to prevent the additional caching at downstream C-routers. (3) The rest of chunks of the file will be transferred to $H1$ without any caching. (4) Suppose another request for the same file from $H2$ is routed towards the original server. Note that the first chunk will be served from C-router A while others will be served by the original server. (5) C-router A will mark the first chunk to be cached at the downstream C-router B, and (6) the original server marks the next 2 chunks (i.e., exponentially increasing number of chunks) to make them cached at C-router A. The chunk 1 (or 2 and 3) will be cached at C-router B (or C-router A) and forwarded to $H2$. The *cache suggestion* bits of the three chunks will be reset to 0, respectively. (7) The rest of the chunks will be forwarded to $H2$ without caching. (8) The third request from $H3$ will be served from C-router B (i.e., the first chunk), C-router A (i.e., the 2nd and 3rd chunks), and the original server (i.e., the remainder). During the file transfer to $H3$, the similar caching process will be performed. (9) The first chunk is cached at C-router C; (10) the 2nd and 3rd chunks are cached at C-router B; and (11) the next 4 chunks from the original server are cached at C-router A.

C. Chunk Caching Algorithm

There are three main decisions to make for content caching: what to cache, what to replace, and where to cache. WAVE dynamically adjusts the number of chunks to be cached

Algorithm 1 Chunk Caching Algorithm

1: x: chunk marking window (CMW) base (e.g., 2,3,...)
2: n: chunk marking window (CMW) state (initial value: 0)
3: t: total number of cached chunks
4: $cached$: index of the latest cached chunk at the downstream router (initial value: 0)
5: i: index of the requested chunk
6:
7: **if** $cached < i \leq \sum_{j=0}^{n} x^j$ **then**
8: mark chunk i to be cached
9: $cached \leftarrow i$
10: **else if** $i \leq cached$ **then**
11: mark chunk i to be cached
12: $cached \leftarrow i$
13: $n \leftarrow \lfloor \log_x i \rfloor$
14: **end if**
15:
16: Transfer the requested chunk i
17:
18: **if** $i == t$ **then**
19: $n \leftarrow n + 1$
20: **end if**

depending on the popularity of files. We take a conservative approach in choosing chunks to be cached since the processing overhead of cache update should be reduced.

(1) What to cache (or how to distribute chunks): The chunk caching algorithm of WAVE is described in Algorithm 1. The number of chunks to be cached (at the downstream C-router of a given router) is determined by the chunk marking window (CMW), which exponentially increases as the number of requests for the file increases. The CMW size is determined by: (i) a base x that determines the speed of chunk caching, and (ii) the current state n (n is called CMW state). Thus, given x and n, the current CMW ranges from $\sum_{j=0}^{n-1} x^j + 1$ to $\sum_{j=0}^{n} x^j$. The CMW state, n, increases by 1 if all of the cached chunks are transferred. As the CMW will be small if the requests (of a file) are not frequent, WAVE can prevent an unpopular file from being widely distributed. On the other hand, WAVE can distribute a popular file fast with the exponentially increasing CMW size. For this WAVE operation, a C-router maintains two counters for a file for each router interface: CMW state n, and the index of the latest chunk whose $cache\ suggestion$ bit is set. We use a variable $cached$ for the latter counter. Thus, the number of total variables in a C-router is $2 \times the\ number\ of\ cached\ files \times the\ number\ of\ interfaces$. Note that WAVE maintains the variables for each file, not for each chunk.

When a C-router (or the original server) caching the chunks receives a request for chunk i through a router interface, it first checks whether the request chunk index i is within the CMW range (line 7). If so, chunk i will be marked to suggest its downstream C-router to cache the chunk (line 8) and the C-router will update $cached$ (line 9). After that, the requested chunk i will be forwarded to the downstream C-router (line 16). After forwarding all the cached chunks, the C-router will increment n by 1 to increase the CMW size exponentially (lines 18-20). If the downstream C-router receives a marked chunk, it will cache the chunk, reset the $cache\ suggestion$ bit, and forward it towards the end host.

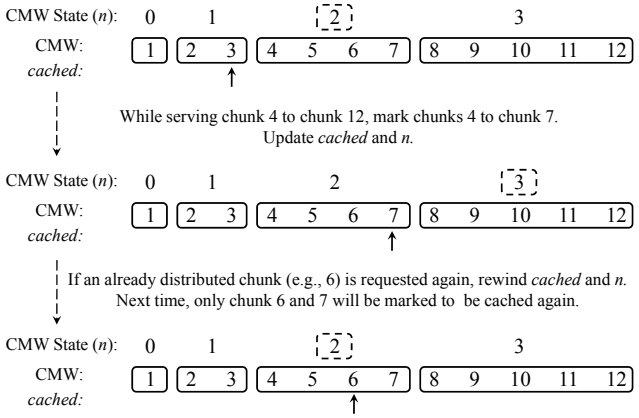


Fig. 2. Content caching example in a C-router.

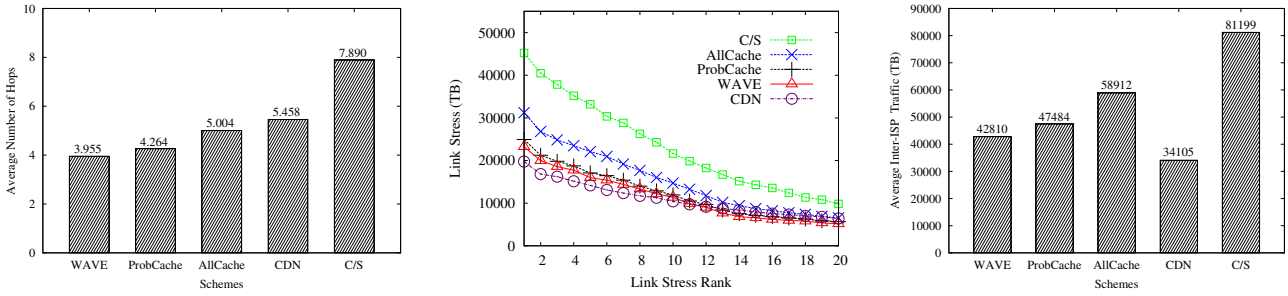
Since C-routers perform the cache replacement independently, it is possible for any of the network-wide distributed chunks to be replaced at any downstream C-router. Therefore, a replaced chunk may be requested again (line 10). In this case, the C-router will mark the chunk to suggest it to be cached again (line 11). Also, $cached$ falls back to i (line 12) and the CMW state n will be modified to $\lfloor \log_x i \rfloor$ (line 13). Note that the requested chunk index i is included in the updated CMW.

The chunk caching algorithm is illustrated in Fig. 2. We assume the file consists of 12 chunks, and the CMW base is 2. Suppose that the given C-router caches all the chunks; the current CMW state n is 2, and $cached$ is 3. While transferring chunks (starting from index 4 to index 12) to its downstream C-router[2], the C-router will mark the chunks in the CMW (chunks 4-7). Then, the chunks 4-7 will be cached at the downstream C-router. Now, $cached$ and n are set to 7 and 3, respectively. If a chunk that has already been distributed (say, chunk 6) is requested again, $cached$ and n are set to 6 and 2(=$\lfloor \log_2 6 \rfloor$). Therefore, chunks 6 and 7 within the current CMW will be marked and forwarded again.

(2) What to replace: When a cache is full and a new chunk to be cached arrives, a victim chunk should be chosen to be replaced. WAVE uses the least recently used (LRU) approach, the most representative cache replacement algorithm used in cache networks [7], [16]. Note that WAVE can use other cache replacement algorithms such as least frequently used (LFU) or least recently/frequently used (LRFU). Since the caching unit of WAVE is a chunk, there would be a huge overhead if the access history information of individual chunks is maintained. Therefore, WAVE maintains the access history in the unit of a file to find a victim chunk to be replaced. When the last cached chunk is replaced, the access history for the file can be removed.

(3) Where to cache: Since there can be a large volume of

[2]Chunks from index 1 to index 3 have already been transferred to its downstream C-routers towards the soliciting host.



(a) Average hop count (b) Link stress (c) Inter-ISP traffic reduction

Fig. 3. Performance comparison of WAVE against ProbCache, AllCache, CDN, and client-server.

in-network storages, it is important to decide where to cache the files. The *direction* and *location* for chunk caching should be carefully considered. WAVE distributes the chunks in the *direction* from which the chunk requests come, considering the spatial locality. Regarding the *location* to distribute the chunks, there can be a few choices: one-hop distribution (i.e., only the next hop caches the chunk), multi-hop distribution (i.e., a chunk is cached after traversing half of the remaining hops), ISP crossing distribution (i.e., a chunk is cached after the boundary of domains is crossed). Basically, WAVE distributes chunks in a hop-by-hop manner to fully utilize in-network storages. The performance of WAVE with different distribution options is omitted due to the space limitation.

III. Simulation Results

A. Simulation Environments

To evaluate WAVE, we conducted simulations using a discrete event-driven simulator. GT-ITM [19] is used to generate a network topology that consists of 1 transit domain and 5 stub domains. There are 5 and 10 C-routers in a transit domain and a stub domain, respectively. Total 1,000 end hosts are connected to 50 C-routers in stub domains. Also, 10 original servers are co-located with randomly selected C-routers, and 100,000 files are randomly distributed at the original servers. Thus, each original server stores 10,000 files, on average. Every content is 1 GB size and each content is divided into 100 chunks. The probability distribution of requests of 100,000 files follows Zipf distribution with parameter 0.85. In our simulations, the default storage size of every C-router is set to 10 GB for content caching.

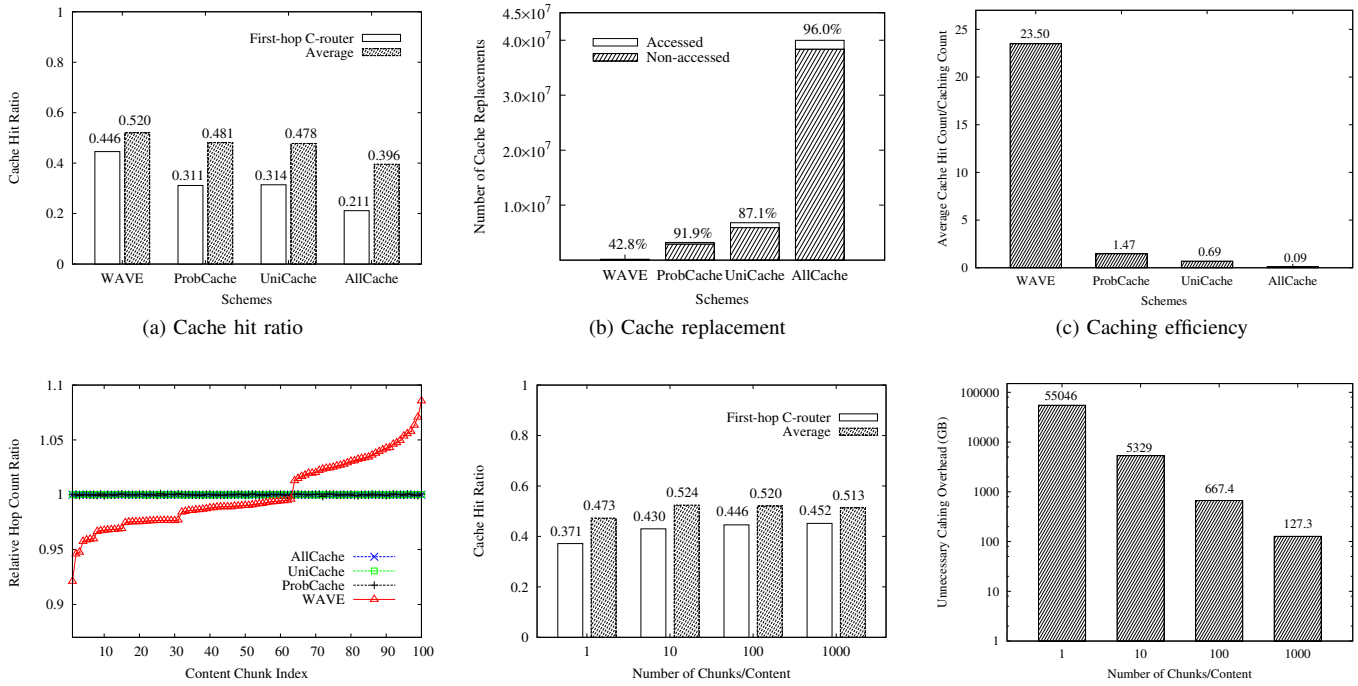
We compare WAVE with other on-demand caching schemes, AllCache, UniCache, and ProbCache. In AllCache, content chunks being delivered are blindly cached by all the C-routers between the end host and the original server. AllCache represents a scheme simply adopting a cache replacement algorithm without carefully considering what to cache. UniCache caches a content chunk at one of the C-routers along the path; thus, the caching probability at each router is $1/hop\ count$. In ProbCache, a content chunk is cached with a fixed probability by each C-router between the host and the original server. (The probability is set to 0.1 since the performance of ProbCache is best with that value in our simulations.) For cache replacement, two well-known algorithms, LRU and LFU are used in all schemes. We only present the results with LRU since LFU shows the similar trend with LRU. Also, we compare WAVE with the client-server and CDN schemes. In the CDN scheme, the request is routed to the closest CDN server. For the CDN server deployment, a CDN server is deployed at the best position in each stub domain in terms of hop count. The storage size of a CDN server is the same as the sum of in-network storage in a single stub domain.

B. Network-wide Performance

We evaluate the performance of the above in-network caching schemes in terms of the average hop count, link stress, and inter-ISP traffic. We omit the results of UniCache since UniCache shows similar performances with ProbCache.

(1) Average Hop Count: As shown in Fig. 3a, the in-network caching schemes including WAVE reduce the average hop count between a host and a content-holding place (either the original server or a C-router that caches the content files) than the client-server model, resulting in faster content retrieval. That is, in-network caching schemes can cache the content files closer to end hosts than the original server. For the popular content, the cached place (i.e., C-router) can be even closer to end hosts than the CDN server. As the average hop count is reduced, the in-network caching schemes can reduce the total traffic volume per ISP. Among the in-network caching schemes, WAVE achieves the shortest average hop count than other schemes thanks to its popularity-based caching. Since AllCache blindly caches all content chunks passing through, it may replace popular chunks with unpopular ones which yields poor performance than WAVE and ProbCache. Therefore, in the content-oriented networks, adopting any cache replacement algorithm without careful consideration for what to cache may lead to sub-optimal performances.

(2) Link Stress: To show the traffic mitigation of in-network caching schemes, we measure the link stress, which is defined as the traffic amount transferred over a particular link, and plot top 20 links in a descending order. As shown in Fig. 3b, the in-network caching schemes use the links in a more load-balanced fashion than the client-server model. Since the CDN servers that store most popular files are placed at the center of each stub domain, the CDN can achieve the lowest link



(a) Cache hit ratio (b) Cache replacement (c) Caching efficiency

(d) Relative hop count ratio (e) Cache hit ratio: chunk size (f) Unnecessary caching overhead: chunk size

Fig. 4. Cache performance comparison of WAVE against ProbCache, UniCache, and AllCache model.

stress performance. We can see that WAVE and ProbCache can distribute the content files to multiple C-routers, resulting in the comparable link stress performance to the CDN.

(3) Inter-ISP Traffic Reduction: The CDN achieves the minimum inter-ISP traffic in our simulation since the top popular files are stored in the CDN servers in advance. Note that in-network caching schemes have to download files at least once if the original servers reside in other ISPs. Overall, the in-network caching schemes can reduce the inter-ISP traffic significantly compared to the client-server model. Since WAVE can cache popular content files more than any other in-network caching schemes, WAVE incurs the lowest average inter-ISP traffic volume among them. AllCache incurs more content downloads from the original servers outside the ISP and hence more average inter-ISP traffic than WAVE and ProbCache since popular content files may be replaced by the unpopular ones due to its popularity-blind caching.

C. Cache-related Performance

In this section, we compare WAVE with the other in-network caching schemes in terms of cache-related performance (e.g., cache hit ratio, cache replacement count, etc). The caching operations such as chunk distribution and cache replacement can affect the overall performance of in-network caching schemes since they need some processing overhead such as memory access and computations (e.g., selecting which content/chunk needs to be replaced).

(1) Cache Hit Ratio: Fig. 4a shows the cache hit ratio (in terms of chunks) of the caching schemes. We plotted the cache hit ratio at the first-hop C-router (white bar) and the average cache hit ratio over all C-routers in the networks (shaded bar). WAVE achieves the highest cache hit ratio than the other schemes by caching the popular chunks more (i.e., at least 13.2% higher hit ratio at the first-hop C-router and 3.9% higher on average). ProbCache and UniCache achieve lower cache hit ratio than WAVE since each caching decision is made with a fixed probability which cannot reflects the popularity of content files. Similarly, AllCache shows the lowest cache hit ratio due to its popularity-blind and aggressive caching.

(2) Cache Replacement Count: To show the efficiency of the cache management, we measure the number of cache replacements as shown in Fig. 4b. WAVE incurs negligible cache replacements due to the efficient caching considering the popularity. Meanwhile, ProbCache and UniCache incur more frequent cache replacements than WAVE since they always cache a chunk with a certain probability. ProbCache shows better performance than Unicache due to the best caching probability it chooses. Similarly, since chunks are always cached at all the C-routers along the path, AllCache incurs the most frequent cache replacements.

We further classify the replaced chunks into two groups depending on whether they have been accessed after caching or not: accessed and non-accessed ones. As shown in Fig. 4b, less than half (42.8%) of chunks are replaced without being accessed in WAVE due to its popularity-based chunk caching algorithm. On the other hand, more than 87% of content chunks are not accessed before being replaced in the other schemes. In particular, AllCache exhibits a vast majority of non-accessed chunks due to its popularity-blind operations.

(3) Caching Efficiency: The caching efficiency of in-network caching schemes is shown in Fig. 4c, which is defined as the average number of cache hit counts divided by the number of caching events (i.e., this is incremented whenever a chunk is cached). It represents how many times a chunk will be used after it is cached. In WAVE, a chunk will be used 23.5 times on average once it is cached, which is at least 16 times higher than other schemes. ProbCache shows better caching efficiency performance than Unicache since ProbCache chooses the best caching probability in our simulations. Also, AllCache shows the lowest caching efficiency performance (0.09 times on average) among the compared schemes.

(4) Relative Hop Count: Recall that in WAVE, the chunks with the smaller indices are distributed before the ones with the larger indices. To see how chunks are distributed (towards the end hosts) depending on their indices, we measure the relative hop count, which is defined as the ratio of how many hops a chunk is distant from the end hosts on average depending on its index to the average hop count (shown in Fig. 3a). As shown in Fig. 4d, WAVE distributes the chunks with the lower indices (up to around index 60) closer to the end hosts than the chunks with the higher indices. For the first chunk, WAVE makes it cached about 8% closer than the other chunks on average and 15% closer than the last chunk. On the other hand, chunks are cached almost at the same distance regardless of the chunk indices in the other schemes. Since the preceding chunks are cached nearer to the end hosts than the following ones, WAVE may be more amenable to supporting sequential delivery for multimedia data.

(5) Number of Chunks: We vary the number of chunks that constitute a file and measure the impact of the number of chunks on the cache hit ratio. As shown in Fig. 4e where x-axis is the number of chunks that constitute a file, the case of one chunk (i.e., 1 chunk is equal to a file) shows the lower cache hit ratio than the cases of other chunk sizes. Note that the case of one chunk is the same as leave copy down (LCD) scheme in the hierarchical web caches [16]. If a file consists of a single chunk, the replacement penalty is high when the popular file is replaced by the unpopular one, resulting in the lowest cache hit ratio. On the other hand, as the chunk sizes becomes smaller, the penalty is reduced since the number of chunks to be replaced will be much smaller than the size of a whole content file. In our simulation environments, chunk size variations (10, 100, 1000 chunks constitute a file respectively) exhibit similar cache hit ratio.

Fig. 4f shows the unnecessary caching overhead which is defined as the total volume of non-accessed chunks. As the number of chunks becomes larger, the unnecessary caching overhead decreases. Therefore, increasing the number of chunks constituting a file will reduce the unnecessary caching overhead. However, increasing the number of chunks necessarily accompanies other overhead: control overhead (i.e., many request packets), transfer overhead (i.e., the portion of header increases), and cache lookup overhead (i.e., the amount of index data).

IV. Conclusion

In this paper, we proposed WAVE for efficient caching and delivery of content. To reflect the content popularity, WAVE exponentially increases the number of chunks of a file to be cached as its request increases. WAVE achieves higher cache hit ratio and less frequent cache replacements than other on-demand caching schemes. We will extend WAVE to support random seeking for multimedia files and analyze performance with different chunk sizes and various distribution options. Also, multi-source/multi-path extension will further be investigated.

Acknowledgment

This research was supported in part by the IT R&D program of KCA(11-913-05-002: Fundamental Research on In-network Caching and Routing for Named-data Networking) and in part by the IT R&D program of KCA(10913-05004: Study on Architecture of Future Internet to Support Mobile Environments and Network Diversity).

References

[1] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang, "Cooperative content distribution and traffic engineering in an ISP network," *Proc. SIGMETRICS*, June 2009.

[2] T. Koponen et al, "A data-oriented (and beyond) network architecture," *Proc. ACM SIGCOMM*, Aug. 2007.

[3] V. Jacobson et al, "Networking named content," *Proc. ACM CoNEXT*, Dec. 2009.

[4] A. Zahemszky, A. Csaszar, P. Nikander, and C. Esteve, "Exploring the pub/sub routing&forwarding space," *Proc. ICC Workshop Future-Net*, June 2009.

[5] K. Cho et al, "How can an ISP merge with a CDN?," *IEEE Communications Magazine*, Oct. 2011.

[6] M. Lee, K. Cho, K. Park, T. T. Kwon, and Y. Choi, "SCAN: scalable content routing for content-aware networking," *in Proc. IEEE ICC*, June 2011.

[7] E. J. Rosensweig and J. Kurose, "Breadcrumbs: efficient, best-effort content location in cache networks," *Proc. INFOCOM*, April 2009.

[8] S. Arianfar, T. Koponen, S. Shenker, and B. Raghavan, "On Preserving Privacy in Information-Centric Networks," *ACM SIGCOMM ICN Workshop*, 2011.

[9] V. Jacobson et al, "VoCCN: voice over content-centric networks," *Proc. ACM ReArch Workshop*, Dec. 2009.

[10] Z. Zhu, S. Wang, X. Yang, V. Jacobson, and L. Zhang, "ACT: Audio Conference Tool Over Named Data Networks," *ACM SIGCOMM ICN Workshop*, 2011.

[11] G. Carofiglio, V. Gehlen, and D. Perino, "Experimental Evaluation of Memory Management in Content-Centric Networking," *IEEE ICC*, 2011.

[12] M. Diallo, S. Fdida, V. Sourlas, P. Flegkas, and L. Tassiulas, "Leveraging caching for Internet-scale content-based publish/subscribe networks," *IEEE ICC*, 2011.

[13] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," *Proc. IEEE INFOCOM*, 2001.

[14] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," *Proc. USENIX ATEC*, 1996.

[15] Sem Borst, Varun Gupta, and Anwar Walid, "Distributed caching algorithm for content distribution networks," *Proc. INFOCOM*, 2010.

[16] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta algorithms for hierarchical web caches," *Proc. IEEE IPCCC*, 2004.

[17] X. Tang and S. T. Chanson, "Coordinated en-route web caching," *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 595-607, 2002.

[18] A. Ghodsi et al, "Information-Centric Networking: Seeing the Forest for the Trees," *Proc. ACM HotNets-X*, 2011.

[19] E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," *Proc. IEEE INFOCOM*, 1996.