ISBN 978-981-11-7861-0
Proceedings of 2018 the 8th International Workshop on Computer Science and Engineering (WCSE 2018)
Bangkok, 28-30 June, 2018, pp. 453-460
doi: 10.18178/wcse.2018.06.079

A Lightweight Cache Insertion Filtering Scheme for Information-Centric Networking

Li Ding [1], Jinlin Wang [2+], Lingfang Wang[2] and Yiqiang Sheng [2]

[1] Department of Automation, University of Science and Technology of China, Hefei, China

[2] National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing, China

Abstract. The success of Information-Centric Networking owes much to the feasibility of large caches. As such, solid state disk (SSD) is widely adopted as a cache device to scale up the cache size up to terabytes in some recent works. Nevertheless, the limited lifetime is a significant drawback of SSD, making it non-trivial to design a cache system involving SSD. Hence, writing operations on SSD should be elaborately considered in order to reduce the frequency of writes without compromising cache hit ratio. This may seem contradictory, but it is possible since the typical Internet traffic patterns follow the Zipf distribution. This characteristic can be fully exploited to store a small set of popular contents. For this reason, this paper proposes a lightweight cache insertion filtering scheme based on least recently used (LRU) queue and hash table. Our goal is to prevent these unpopular contents from entering into the cache and only store high frequently requested contents. The experimental results show that our proposed scheme can reduce the number of writing operations on SSD without comprising cache hit ratio. In fact, it can improve cache hit, compared with the unfiltering scheme and a lightweight probationary insertion filtering scheme, while only requiring a modest memory and consuming about two hundred CPU cycles.

Keywords: Information-Centric Networking, SSD, filtering, cache hit ratio, CPU cycle.

1. Introduction

Information-Centric Networking (ICN) [1] emerges as a promising future architecture to address the problem of content distribution efficiently. One of the main characteristics of ICN is in-network caching. As ICN conceives caching at the network layer making it one of the core functionalities [2], a router is thereby required to be able to perform wire speed caching also. However, current single storage technology can not reach a tradeoff on high storage capacity and high speed [3], [4]. For example, DRAM technology can sustain O(10 Gbps) speed, but it can only provide O(10 GB) capacity. On the contrary, technologies (such as SSD) that can provide O(1 TB) size is not able to meet the requirements of O(10 Gbps) line speed. In this context, hierarchical/hybrid storage system is a potential candidate that can realize both goals. In fact, some recent works have proposed a DRAM and SSD based two-level cache [5], [6], [7]. When designing such a system, there are two factors needed to be considered carefully.

On the one hand, it is important to adopt an efficient cache management policy due to the existence of underlying SSD. Unlike DRAM, SSD suffers from limited lifetime due to the fact that NAND flash can only be programmed or erased for limited times [8]. For instance, the endurance of a typical SSD product is about 10K to 100K writes [9]. The limited lifetime of SSDs hinders their deployment in reliability sensitive environments. Therefore, in order to improve the utility of SSD, one needs to carefully design a cache management strategy to minimise the amount of writing operations on SSD. But, such a write reduction

[+] Corresponding author.
E-mail address: wangjl@dsp.ac.cn.

needs to be achieved without compromising cache hit ratio of the system, since reducing the frequency of writes to SSD may decrease the number of distinct content items cached on the router and affect hit ratio.

On the other hand, typical Web content retrieval workloads are characterised by a high fraction of contents which are requested only once over a long period of time. For example, the authors reported that in the web traffic served by their CDN server, 74% of the roughly 400 million objects in cache were accessed only once [10]. In fact, only a small set of popular contents is frequently requested. Caching contents that will be requested only once can bring no benefit as they will never be requested again. Moreover, these contents can bring extra processing pressure on ICN routers due to frequent cache insert, lookup or replace operations if they are stored without any filtering technology.

Given all this, it is necessary to design an efficient packet filtering scheme. It should filter these contents with low frequency of request to leave space for high frequently requested contents, and reduce the number of writing operations on SSD. More importantly, this filtering scheme should not be realized at the cost of sacrificing cache hit, which is one of the most important performance metrics for cache system.

The rest of the paper is organized as follows. We first review some related works in Section 2, and then describe our proposed filtering scheme in section 3. Our experimental results are shown in section 4. The article closes with a conclusion and a future work in Section 5.

2. Related Work

Recently, packet filtering schemes have been proposed directly for hierarchical cache systems, which combine the advantages of DRAM and SSD to meet high storage capacity and high speed [7], [12]. Considering the inherent characteristics of SSD technology, they both point out that one needs to carefully design an efficient cache management policy at the DRAM level to reduce the number of writing operations on SSD. However, such a write reduction needs to be achieved without comprising the cache hit since this may impact the freshness of stored contents. Besides, as caching in ICN is required to be performed at wire speed, this cache management policy should be lightweight in order not to affect caching operations.

Specifically, Jiang et al. [7] proposed a filtering scheme called uCache, which uses two LRU queues to filter unpopular contents. The first level queue stores items that enter into the DRAM cache the first time. When a content item gets a cache hit in this level, uCache moves this item to the second level queue rather than moving it to the head of LRU queue as a traditional LRU queue does. Consequently, items residing in the second level queue are re-accessed at least once or even more, and they can be written into SSD cache. Saino [12] proposed a lightweight probationary insertion filtering scheme. According to this scheme, each item in DRAM is associated with a counter. If this item is requested when in DRAM, this counter is increased by one and the content is moved to the top of the LRU queue. When the content reaches the bottom of the LRU queue, this scheme will evaluate this counter to determine whether pushing this item into SSD or not. If the counter of this item is bigger than the predefined threshold, it can be inserted into SSD, otherwise it is discarded. The drawback of both schemes is that unpopular contents can not be prevented from entering into DRAM.

In [10] the authors also considered using bloom filter to filter one-timers contents. When a caching node receives a request each time, the bloom filter is consulted to judge whether there is a record. If there exists a record in this bloom filter, this content can be cached. Otherwise, this content is added to the bloom filter and it is discarded. However, a lookup/insert operation requires a few hash computations and load/store operations in random areas of the Bloom filters. Therefore, it is not suitable for line-speed operations required by ICN routers.

3. Proposed Filtering Scheme

In this section, we first show our motivation and then design the data structures used to filter contents. Afterwards, we describe two key processing procedures in this scheme. Finally, we analyze the time complexity and memory consumption of our proposed scheme. For brevity, notations mentioned in this paper are summarized in Table 1.

3.1. Motivation

By analyzing an one-hour Internet trace from Wikipedia [11] on Sep 20, 2007, which comprises ~5.4 million requests over a catalogue of ~1.2 million content items, we find that 0.84 million content items are only requested once, which occupies 72.5% of the total contents. Moreover, more than 90% contents are requested no more than three times. Due to this fact, caching these contents on content store brings less benefit. It is expected to filter these contents that are requested less frequently out to leave space for more frequently requested contents.

Table 1: Summary of key notations

N_{hash}	the size of hash table	N_{bucket}	the number of entry a bucket contains	L_{lru}	the length of LRU queue
T_{req}	minimum request times for passing the filter	Q_{lru}	the LRU queue	T_{hash}	the hash table

3.2. Data structures

There are two problems to be considered when designing an efficient filter. First, the limited storage space determines that this filter can not consume too much memory. Second, this filter should be performed quickly so as not to push extra pressure on ICN routers. To filter contents efficiently, the key is to identify contents with a low frequency of request. Since users in ICN issue an Interest packet to request a content and popular contents will be requested frequently along the time, the number of Interest packets received by a router within a certain time window can be used as a representation of request frequency. When the corresponding content returns, the router can get the request frequency from saved information of previously received Interest packets.

With this in mind, we decide to adopt a LRU queue and hash table to implement a filter. The LRU queue, which has a limited length, is used to control the statistical period and cache replacement when the queue is full. Its structure is shown in Fig. 1. A hash table is used to record the requested content and the corresponding request times, which is shown Fig. 1 also. Each entry in this hash table includes four fields, a flag field which indicates whether the entry is valid or not, a request times field which indicates the request times of this content, a hash value field which is a 32-bit hash value of the requested content, and a index field which stores a pointer to the location in LRU queue. There are two goals for designing such a hash table.

Our first goal is to minimize memory access latency. For this purpose, the size of each bucket is fixed to one cache line, i.e., 64 bytes. In such a way, even if the bucket is not in CPU cache, it can be retrieved with a single memory read operation. This is very useful in case of collisions since it is possible to locate the desired entry just by iterating the bucket, which is in L1 cache and can be accessed very quickly. Bucket overflow can still occur if there are more than four colliding entries. In this case, we can use open addressing method for collision resolution, but it is expected to be rare if we choose a good hash function and configure a hash table with a large enough number of buckets. In order to speed up lookup operations, each entry in this hash table has a flag to indicate whether this entry is occupied or not. Hash value comparison will only be performed on an entry which is occupied.

Our second design goal is to reduce storage overhead. As a result, each entry only stores the hash value of the requested content rather than the variable name of the requested content. This can speed up the lookup operation when a content packet returns. Considering that different content names may be mapped to the same hash value, an unrequested content will have a non-zero request times value. Consequently, this content may pass the filter. This influence can be solved by choosing a hash function with a low collision probability. Besides, as the length of the LRU queue is limited, if a content is not requested within this time window, it will be replaced by a requested content. The request times of this unrequested content will decrease by one, and finally have zero values (In this case, the busy flag of this entry is set to be invalid, see next section). Hence, hash collision can be alleviated. Packets with a low frequency of request can still be filtered effectively.

We adopt the LRU queue rather than FIFO queue since it considers the recency of requests. For this reason, if a content is frequently requested with the time window, it will have a high probability to be moved to the head of the queue. On the contrary, unpopular contents which are requested less frequently are prone to be closer at the end of the queue. Hence, they are more likely to be replaced. Once replaced, the request

times field of the corresponding entry in the hash table will be decreased by one (see next section), which reduces the probability of this content to be cached. Therefore, these two simple data structures are effective to filter unpopular contents.

3.3. Processing procedures

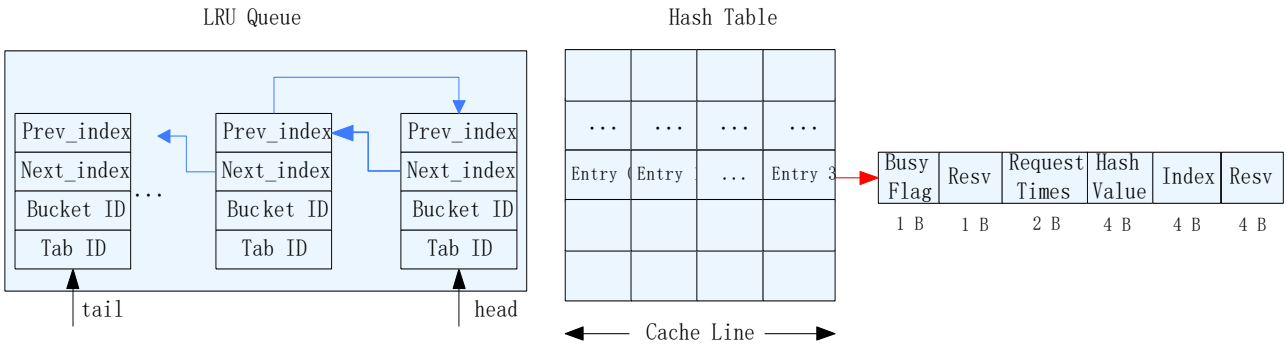


Fig. 1: Data structures of Hash Table and LRU Queue.

In this section, we describe how our proposed filtering scheme can be applied in an ICN router. The key processing procedures include Interest Recording (IR) and Content Filtering (CF) stages. Specifically, if the router receives an Interest packet, it will first lookup the hash table to judge whether receiving the same Interest packet before. If the router finds a matching entry in the hash table (i.e., the hash value of the name of the requested content is equal to the hash value recorded in this entry), the request times field in this entry will be increased by one. At the same time, this item will be moved to the head of LRU queue. Otherwise, the router will iterate this bucket to find an available entry and store the hash value in it. When inserting a new item in the hash table, if the LRU queue is full, an item in this queue will be replaced. Meanwhile, the request times field of the entry recording this replaced item will decrease by one. Once the request times field is equal to zero, this entry is marked invalid, which can be used for storing information of subsequent Interest packets. If the router receives a content, the router will lookup the hash table to get the matching entry. If an entry is found, then the router checks whether the request times field in this entry is larger or equal to the predefined threshold Treq. If so, this content packet can be cached on this router. Otherwise, this content is thought not to be popular enough. Hence, it will be discarded. If no entry is found, which means the content is not requested before, in this case this content will definitely not be cached.

3.4. Characteristics

Now we analyze the time complexity of our proposed scheme. As a router in ICN will compute the hash value of content name before performing lookup operation on a content store, this hash value can just be as the input hash value of our filter. It is not necessary to recompute hash value again. Hence, CF stage will only need to iterate the corresponding bucket once to judge whether the request times field in this entry is larger than the predefined threshold or not, which can be fast as one read operation will copy the whole bucket into the CPU L1 cache. The complexity of this processing stage is O(1). Compared with CF stage, IR stage needs to iterate the bucket one more times for a new Interest, which also has an O(1) complexity. Besides, IR stage requires updating LRU queue. But this operation is independent with the length of LRU queue, which has an O(1) complexity. Overall, both CF and IR stages have an O(1) time complexity. In the experiments later in section 4.2, we can find that the increased CPU cycles brought by our filtering scheme is less than 1% for the total cycles without any filtering scheme. Hence, our proposed scheme is sufficiently lightweight and will not decrease packet forwarding rate.

As for the memory consumption required by the LRU queue and hash table, they can be computed according to equation $M = L_{lru} * 16 + N_{hash} * 64$. Considering that a popular content item will be frequently requested, counting the request times of one content within a certain time window is reasonable to judge whether it is popular or not. Hence, L_{lru} can have a limited value. As each item in Q_{lru} has a corresponding entry in T_{hash}, a large N_{hash} value is required to reduce collision probability. In this case, L_{lru} is much smaller

than N_{bucket}. Therefore, the main consumption is determined by the hash table. Specially, for a hash table with one million buckets, which can store four million content items ideally, our proposed scheme only requires a modest memory overhead (68.66 MB).

4. Experiments

In this section, we first present the experimental setup. Then we measure the number of CPU cycles needed for the execution of IR and CF stage. Finally, we perform the evaluations to demonstrate the advantage of our proposed scheme compared with the unfiltering scheme and a probationary insertion scheme [12].

4.1. Experimental Setup

We implement the hierarchical DRAM and SSD based content store prototype [12] on a server equipped with two Intel Xeon E5-2620 hex-core CPUs as it can enable 20 Gbps speeds based on inexpensive commodity hardware. We connect this server to a traffic generator equipped with two 10 Gbps optical interfaces. Interest packets are originated at the traffic generator from port0 and transmitted to the router. If an Interest packet gets a hit, the router will respond a data packet. Otherwise, it will be forwarded out from port1. The traffic generator will reply to the incoming Interest packets with data packets matching the Interests' content name, and this data packet will finally be forwarded from port0. As our focus is on the performance of cache hit ratio, the router does not involve the PIT and FIB related operations.

As evaluation workload we use two kinds of traces with different popularity distribution, a generated trace and a real Internet trace. The characteristics of these two traces are shown in TABLE 2. Besides, the difference between the generated trace and the Wikipedia trace is that they have different name length. In the generated trace, the content item is the ranking index of this content. However, the Wikipedia trace has variable length URLs, which have a longer length. We use such two traces since they can represent two contrasting laws: a normal popularity distribution and a more accentuated popularity distribution. Each content item contains eight chunks. Other configurations are similar to the settings in [12]. We use the first 25% of requests to warm up caches and then measure performance over all requests.

Table 2: Traffic traces

Trace type	Total request number	Distinct content items	Zipf exponent α	Duration
Generated trace	5396479	1156983	0.96	1h
Wikipedia trace	5396479	956145	0.79	-

4.2. CPU Cycle

In this section, we measure the number of CPU cycles required by two key processing stages, i.e., IR and CF. These two stages are independent with the underlying cache. For simplicity, we only enable DRAM cache and use the first 25% of requests to get an average value. Results are shown in Fig. 2. For the purpose of comparison, we also show the operation that requires the most CPU cycles when processing an Interest and Data packet according to [13]. For Wikipedia trace, we can find that the number of cycles required by IR stage (about 180 cycles) is about 19.4% of FIB lookup operation. CF consumes about 35 cycles, which is only 4.24% of CS lookup and insert operation. Moreover, if we consider all necessary operations, the average CPU cycles consumed for processing a pair of Interest and Data packets are C = 2794.2 cycles [13]. However, the increased cycles brought by our filter are about 215 cycles, which is less than 1% for the total cycles. Therefore, our proposed scheme will have little impact on the packet processing in ICN routers. Similar results can be found for the generated trace. Although the generated trace and Wikipedia trace have different characteristic, these two stages consume nearly the same CPU cycles. This is expected since both IR stage and CF stage process the hash value of a content item, rather than the variable name.

4.3. Performance on Single-Layer Cache System

Before evaluating the performance on the hierarchical system, we examine the impact of Llru and Treq on the single-layer cache system with DRAM cache only. In this case, the ratio between content catalogue and DRAM size is equal to 100. We first measure the cache hit ratio of the system without any filtering

technology. Then we apply our proposed algorithm and get the cache hit ratio. The experimental results are shown in Fig. 3 and 4.

There are two important observations we can make from this figure. First, different Llru values show the similar trend as Treq varies. Increasing the size of Treq can also bring a larger cache hit when Treq is less than a threshold (25 for the generated trace, 10 for the Wikipedia trace). This is expected as contents with a low frequency of request are filtered and only high frequently contents are stored. However, further increasing Treq beyond this value will decrease the cache hit as the freshness of stored contents is influenced and some contents are prevented from entering into DRAM. Second, a larger Llru does not always result in a higher cache hit ratio. Since a larger Llru means that cache replacement can only happen when receiving more requests. During this time window, it is more likely to receive more unpopular contents, which will result in cache replacement. Consequently, the request times of the replaced popular content will be decreased by one (see section 3.3) and may not satisfy the minimum request times threshold.

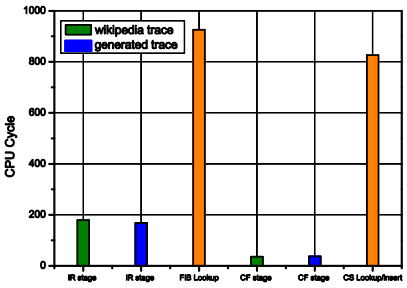


Fig. 2: CPU cycles

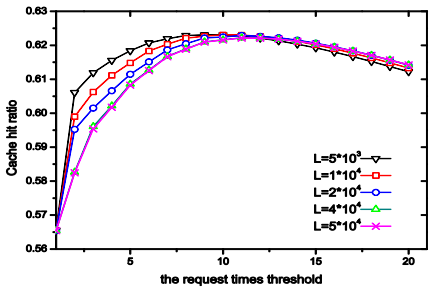


Fig. 3: Cache hit ratio, Wikipedia

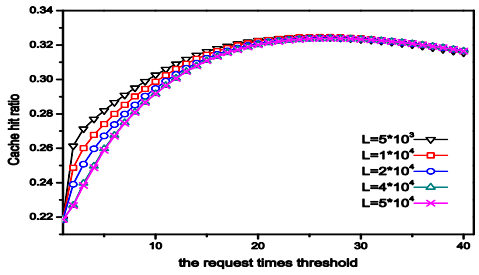
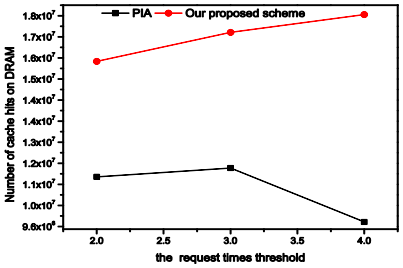
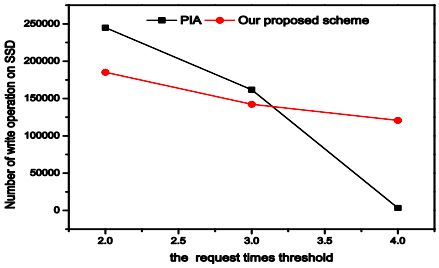


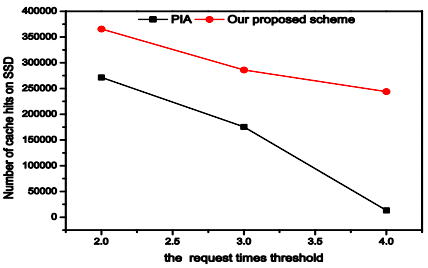
Fig. 4: Cache hit ratio, Generated.



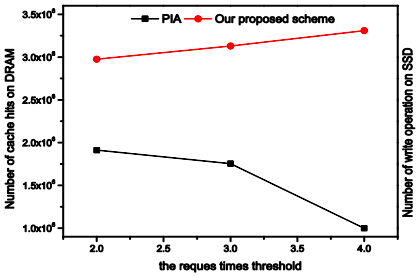
(a) Number of cache hits on DRAM, Wikipdia



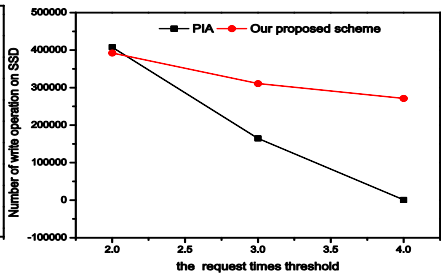
(c) Number of write operation on SSD, Wikipedia



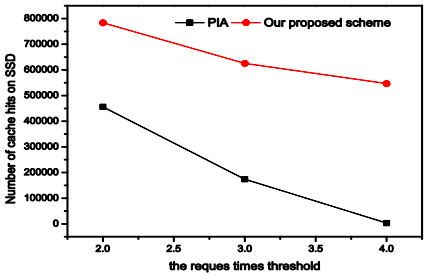
(e) Number of cache hits on SSD, Wikipedia



(b) Number of cache hits on DRAM, Generated



(d) Number of write operation on SSD, Generated



(f) Number of cache hits on SSD, Generated

Fig. 5: Performance evaluations on the hierarchical cache system.

However, if we set a larger Treq, this impact will be alleviated since very popular contents are always frequently requested. The request times of the replaced popular content will be increased to satisfy the threshold requirement due to frequent requests. Hence, the performance gap for different Llru values decreases as Treq increases, as shown in Fig. 3 and 4. Overall, when applying our proposed scheme, the cache hit ratio can be improved significantly for various settings of Llru and Treq when Treq >= 2. The cache hit without filtering and with our proposed scheme is 0.565 and 0.623 for the Wikipedia trace, and 0.218 and 0.324 for the generated trace. We can see that the highest cache hit is increased by 48.6% and 10.27% for the generated trace and Wikipedia trace respectively.

4.4. Performance on Hierarchical Cache System

Now we further evaluate the system performance when enabling both DRAM cache and SSD cache. In this case, the ratio between content catalogue and SSD cache is equal to 100 and the ratio between SSD and DRAM cache is equal to 10. According to the results of single-layer cache system, Llru is set to 5×103 to get the best performance when Treq is less than the optimal threshold value. We will change Treq value to compare the performance of our proposed scheme and the probationary insertion algorithm (PIA) [12].

We get the first look at the number of cache hits on DRAM. As it can be clearly seen from Fig. 5(a) and 5(b), the threshold value that can maximize cache hits for PIA is 3 and 2 for the Wikipedia trace and generated trace, respectively. At this point, our proposed scheme improves the number of cache hits by 43.58% and 55.70% for Wikipedia trace and generated trace correspondingly. Now we analyze the performance metrics on SSD when the threshold value has the aforementioned value, which are shown in Fig. 5(c), 5(d), 5(e) and 5(f). The most striking result is that our proposed filter can result in a smaller number of writing operations on SSD. However, this reduction does not decrease the number of SSD hits. In contrast, it improves the cache hit at some degree than PIA. This demonstrates that not only packets entering into SSD are required to be filtered, but also packets that enters into DRAM. Our proposed scheme filters packet at DRAM level to ensure that content packets stored in both DRAM and SSD cache are frequently requested. For this reason, it can reduce the number of writing operations without comprising cache hit.

When the threshold value is larger than the aforementioned value, the number of cache hits on DRAM shows a declining trend for PIA on both traces. In this situation, our proposed scheme still results in a larger number of cache hits. As for the number of writing operations on SSD, we can find that the number of segments writing on SSD for PIA is much smaller than that of our proposed scheme. The reason for this can be explained as follows. Since DRAM cache has a limited space, cache replacement will occur once DRAM cache is full. When adopting PIA, the request times of the popular segments may not reach or exceed the threshold during their sojourn time in DRAM when they are replaced by new segments. Consequently, they are thought not to be popular enough and thus will not be inserted into SSD. This situation can be avoided in our proposed scheme as we insert a segment into DRAM only when the request time of it exceeds the threshold. Hence, we do not apply any filter technology at the SSD level and all the replaced segment can be written into SSD cache directly. The increased number of segments written on SSD improves SSD hits significantly, as shown in Fig. 5(e) and 5(f).

5. Conclusion and Future Work

In this paper, we propose a lightweight cache insertion filtering scheme for ICN. Our goal is to filter packets with a low frequency of request and leave space for high frequently requested contents. The experimental results on two traces representing the two different typical popularity distribution verify the effectiveness of our proposed scheme. In the future, we will investigate how to derive the optimal values of the length of LRU queue and the threshold of request times from the characteristics of traces.

6. References

[1] George Xylomenos, Christopher N. Ververidis, Vasilios A. Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Kat-saros, and George C. Polyzos. *A survey of information-centric networking research.* IEEE Communications Surveys and Tutorials, 16(2):1024–1049, 2014.

[2] Anshuman Kalla and Sudhir Kumar Sharma. *A constructive review of in-network caching: A core functionality of*

icn. In International Conference on Computing, Communication and Automation, pages 567–574, 2017.

[3] Somaya Arianfar, Pekka Nikander, and Jörg Ott. *Packet-level caching for information-centric networking.* Acm Sigcomm, 2010.

[4] Diego Perino and Matteo Varvello. *A reality check for content centric networking.* In ACM SIGCOMM Workshop on Information-Centric NETWORKING, pages 44–49, 2011.

[5] G Rossini, D Rossi, M Garetto, and E Leonardi. *Multi-terabyte and multi-gbps information centric routers.* In IEEE 33rd International Conference on Computer Communications, pages 181–189, 2014.

[6] Rodrigo B. Mansilha, Lorenzo Saino, Marinho P. Barcellos, Massimo Gallo, Emilio Leonardi, Diego Perino, and Dario Rossi. *Hierarchical content stores in high-speed icn routers: Emulation and prototype implementation.* In ACM 2nd Conference on Information-Centric Networking, pages 59–68, 2015.

[7] Dejun Jiang, Yukun Che, Jin Xiong, and Xiaosong Ma. ucache: *A utility-aware multilevel ssd cache management policy*. In IEEE 15th International Conference on High Performance Computing and Communications, pages 391–398, 2014.

[8] Guanying Wu and Xubin He. *Delta-ftl:improving ssd lifetime via exploiting content locality.* In Proceedings of the 7th ACM European conference on Computer Systems, pages 253–266, 2012.

[9] Laura M Grupp, Adrian M Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H Siegel, and Jack K Wolf. *Characterizing flash memory: anomalies, observations, and applications.* In IEEE/ACM 42nd International Symposium on Microarchitecture, pages 24–33, 2009.

[10] Bruce M. Maggs and Ramesh K. Sitaraman. *Algorithmic nuggets in content delivery.* Acm Sigcomm Computer Communication Review, 45(3):52–66, 2015.

[11] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. *Wikipedia workload analysis for decentralized hosting.* Computer Networks, 53(11):1830–1845, 2009.

[12] Lorenzo Saino. *On the design of efficient caching systems.* PhD thesis, University College London, 2015.

[13] Kosuke Taniguchi, Junji Takemasa, Yuki Koizumi, and Toru Hasegawa. A *method for designing high-speed software ndn routers.* In ACM 3rd Conference on Information-Centric Networking, pages 203–204, 2016.