Analysis on Caching Large Content for Information Centric Networking

Yoji Yamamoto  
Osaka University  
Suita, Osaka, Japan  
y-yamamoto@ist.osaka-u.ac.jp

Junji Takemasa  
Osaka University  
Suita, Osaka, Japan  
j-takemasa@ist.osaka-u.ac.jp

Yuki Koizumi  
Osaka University  
Suita, Osaka, Japan  
ykoizumi@ist.osaka-u.ac.jp

Toru Hasegawa  
Osaka University  
Suita, Osaka, Japan  
t-hasegawa@ist.osaka-u.ac.jp

ABSTRACT
Increase in the size of content due to the emergence of bandwidth-intensive applications leads to a problematic phenomenon, looped replacement, of packet-level caching of information centric networking. This paper analytically investigates how looped replacement occurs. The analyses imply that cache admission is one of countermeasures against looped replacement. Through simulation, we confirm cache admission suppress looped replacement through simulation.

CCS CONCEPTS
• Networks → Packet-switching networks; In-network processing;

KEYWORDS
Information Centric Networking, Cache, Large Object

ACM Reference Format:

1 INTRODUCTION
Emergence of bandwidth-intensive applications, such as video on demand (VoD), leads to increase in the size of content as well as the volume of Internet traffic. Caching of information centric networking (ICN) [1], which allows routers to cache content, plays an important role to reduce the traffic volume by eliminating redundant content transfer [9, 15, 16]. Furthermore, ICN names each packet, thereby naturally supporting packet-level caching, where content is cached as packets rather than content objects [4]. Packet-level caching is a key to reducing the traffic volume because it eliminates redundant transfer at a finer granularity [2, 3].

Since time complexity is a serious concern of caching, especially for packet-level caching of ICN routers, many existing ICN routers adopt lightweight cache algorithms, such as first in first out (FIFO) and least recently used (LRU) [17]. It is a well-known fact that the time complexity of frequency-based cache replacement algorithms like least frequently used (LFU) is logarithmic in the number of packets in a cache, whereas that of FIFO and LRU is constant [6].

Despite the lightweight computation of LRU and FIFO, storing large objects to an LRU-based or a FIFO-based cache causes a problem, referred to as looped replacement [19]. Looped replacement is a phenomenon that a packet of an object which arrives at a cache evicts another packet of the same object. This phenomenon might waste space of the cache because packets that never hit remain in the cache.

Thomas et al. [19] have developed object-oriented packet caching (OPC), which is a hybrid of packet-level and object-level caching, to cope with problems of packet-level caching, including looped replacement. Although OPC successfully improves the cache hit probability of packet-level caching, it works only on top of architecture having a naming scheme with hierarchical names, such as content centric networking (CCN) [12] and named data networking (NDN) [21]. More precisely, OPC uses objects as its unit of cache management rather than packets. Object names can be obtained by deriving one or more component shorter prefixes of packet names; however, this principle is not applicable to naming schemes that do not adopt hierarchical names, such as tag-based ICN [14] and keyword-based ICN [5]. This paper addresses the issue of looped replacement without assuming the hierarchical naming schemes and develops a countermeasure against looped replacement that is applicable to ICN instances with any kinds of naming schemes.

In the previous study [20], we have developed an analytical model of packet-level caching in preparation for investigating looped replacement. The present paper revisits the issue and investigates impacts of looped replacement on packet-level caching by using the analytical model.

The key contributions of this paper are twofold. First, we analytically investigate how looped replacement occurs and reveal two implications. One is that looped replacement occurs more frequently as the size of objects increases. The other is that not to
admit unpopular objects is one of countermeasures against looped replacement. Second, we demonstrate that cache admission, which decides whether packets arriving at a cache should be inserted into the cache or not, suppresses the occurrence of looped replacement.

The rest of this paper is organized as follows. Section 2 explains packet-level caching and how packet-level caching results in the inefficient utilization of the space of a cache because of looped replacement. Section 3 presents our analytical model of packet-level caching. Analyses on looped replacement with the proposed analytical model are conducted in Section 4. Section 5 demonstrates cache admission suppresses looped replacement. Section 6 briefly overviews related work and Section 7 concludes this paper.

2  PROBLEM FORMULATION

2.1 Object-level and Packet-level Caching

While object-level caching inserts and evicts an entire object at once with a single operation, packet-level caching handles objects on a packet-by-packet basis. That is, an object is divided into several packets and cache insertion and eviction are operated on the individual packets.

Figure 1 is a schematic of packet-level caching. In the figure, a cache is depicted as a queue where the leftmost object/packet is farthest from eviction and the rightmost one is ready to be evicted. The order is determined by cache replacement algorithms, such as LRU, LFU, and FIFO. For instance, in the case of LRU, the leftmost and the rightmost objects/packets in the cache are the most and the least recently requested ones, respectively.

In the case of object-level caching, objects are inserted and evicted at once. When a new object arrives at the cache, one or more objects are evicted from the rightmost side of the cache to make room to insert the arriving object and the arriving object is inserted there. In contrast, in the case of packet-level caching, packets are inserted and evicted individually. Packet-level caching can hold a part of an object. However, this results in inefficient utilization of caches depending on cache replacement algorithms.

2.2 Looped Replacement

Object-level caching is often used at the application layer, such as web caches [8]. In contrast, packet-level caching is mainly operated at the network layer, and time complexity as well as cache hit probability is a serious concern. Many ICN routers, hence, adopt lightweight cache replacement algorithms, such as LRU and FIFO.

LRU and FIFO, however, have a possibility of wasting the space of a cache due to looped replacement. Looped replacement is originally defined by Thomas et al. [19] as a phenomenon that packets of an object are evicted from a cache because of insertion of other packets of the same object. The phenomenon causes cache misses for all the packets of the object even though a part of the object is in the cache. This paper generalizes the definition of looped replacement as a phenomenon that requests to packets of an object result in cache misses even though a part of the object is in the cache.

Looped replacement is caused by the three factors: First, packets of an object are generally requested in the order of its first to last packets. Second, LRU and FIFO evict packets in the same order. Third, there are time gaps between the insertion and the eviction of the first and the last packets in the case of packet-level caching. The time gaps cause a situation where a request to the first packet arrives while some of the packets of the same object is being evicted from the cache. In this case, all the requests to the objects do not hit although some of the packets are in the cache. More precisely, let us consider a situation where some packets of an object have been evicted from a cache, as shown in Fig. 1. When the first packet of the object arrives at the cache, it misses and pushes one of the remainder of the packets out of the cache. The second packet also misses and pushes another packet in the same way. This is repeated until the remainder of the packets are evicted.

3  ANALYTICAL MODEL

3.1 System Model and Assumptions

We consider an ICN router equipped with a cache adopting LRU. A schematic of the cache is shown in Fig. 2. The size of the cache is $C$ slots, each of which stores one packet. Without loss of generality, we assume that packets are sorted in descending order of their recency in the cache. That is, the most recently requested packet is stored at the leftmost side of the cache. The set of all the objects is denoted by $O$. Object $i$ consists of $m_i$ packets, which is referred to as object size. Object $i$ is denoted by an ordered set $P_i = \{p_1^i, \ldots, p_{m_i}^i\}$. The set of all packets is expressed as $N = \bigcup_{i \in O} P_i$.

Packets of each object are requested in the order of their IDs. We assume the independent reference model, i.e., requests to object $i$ arrives according to a Poisson process with rate $\lambda_i$. In this case, the first packet $p_1^i$ also arrives according to the same Poisson process. Its consecutive packets are requested with a constant interval $1/\mu_i$. In the case that object $i$ is a VoD video, $\mu_i$ is determined according to its bit rate.

3.2 Overview

The proposed model is similar to the model proposed by Che et al. [8]. Unlike object-level caching, there are time gaps between the arrival and the eviction of the first and the last packets in the case of packet-level caching. Our model, hence, derives the probability of looped replacement and cache hit by modeling the average interval between insertion and eviction of packets constituting an object and the probability that the next request to the object arrives during
the interval. We refer to probability of cache hit and looped replacement as cache hit probability and looped replacement probability, respectively.

Factors that push a packet toward the rightmost side are different depending on situations where the packet exists in the cache. As shown in Fig. 2, whole the situations can be categorized into the following three states: (State 1) where packets \( p \in \mathcal{P}_1 \) are arriving at the cache, (State 2) where all the packets \( \mathcal{P}_1 \) are in the cache, and (State 3) where packets \( p \in \mathcal{P}_1 \) are being evicted from the cache. The probability \( \pi_i \) that looped replacement of object \( i \) occurs is derived as the probability that a request to \( p_i \) arrives during state 3.

### 3.3 Analytical Model

The terms \( r_1^i \), \( r_2^i \), and \( r_3^i \) denote the expected time when object \( i \) stays at each of the three states. The time between the insertion and the eviction of the first packet \( p_i^1 \) is derived as \( T_F^i = r_1^i + r_2^i \). Similarly, the time between the insertion of \( p_i^1 \) and the eviction of the last packet \( p_i^{m_i} \) is \( T_L^i = r_1^i + r_2^i + r_3^i \).

Looped replacement of object \( i \) occurs if an interval \( t \) between two consecutive requests to \( i \) satisfies \( T_F^i < t < T_L^i \). The looped replacement probability \( \pi_i \) is calculated, by using the cumulative distribution function of an exponential distribution with rate parameter \( \lambda_i \), \( P(t, \lambda_i) \), as follows:

\[
\pi_i = P(T_L^i, \lambda_i) - P(T_F^i, \lambda_i) = e^{-\lambda_i T_L^i} - e^{-\lambda_i T_F^i}.
\]

Since intervals between two packets are constant, \( 1/\mu_i \). \( r_1^i \) is derived as

\[
r_1^i = (m_i - 1)/\mu_i.
\]

While object \( i \) is in state 1, packets of other objects \( p \in \mathcal{N} \setminus \mathcal{P}_1 \), which are shown as gray packets in Fig. 2, also arrive at the cache and they increase the number of packets between \( p_i^1 \) and \( p_i^{m_i} \). A packet of object \( j \) \((j \neq i)\) that arrives at the cache during the arrivals of \( p_i^1 \) and \( p_i^{m_i} \) is referred to as an interrupting packet. The number of interrupting packets of object \( j \) in object \( i \) is denoted by \( a_{ij} \). The increase in the number of interrupting packets leads to decrease in \( r_2^i \) and increase in \( r_3^i \). Since \( a_{ij} \) is the number of packets arriving at the cache within \( r_1^i \), it is obtained as

\[
a_{ij} = m_j P(r_1^i, \lambda_j).
\]

The total number of interrupting packets in object \( i \) is calculated as follows:

\[
a_i = \sum_{j \in \mathcal{O}\backslash\{i\}} a_{ij}.
\]

### 4 ANALYSIS ON LOOPED REPLACEMENT

#### 4.1 Evaluation Condition

This section evaluates the looped replacement probability under the following two scenarios: one is a VoD scenario and the other is a traffic-mix scenario. We evaluate the significance of looped replacement under the VoD scenario and evaluate effects of object size on looped replacement under the traffic-mix scenario.

The VoD scenario focuses on a situation where large objects, such as VoD movies, dominate the whole traffic because the looped replacement probability may increase with the increase in the object size. Focusing on VoD services, such as Netflix, we deploy 18,000 video objects according to the measurement results in [7]. The length of the video objects is set to 60 minutes, assuming TV programs in VoD services. The bit rate is set to 5 Mbps, which is the recommended bit rate for HD quality videos [13]. Objects are divided into 1 KByte packets. In this case, a video object of 60 minutes consists of \( 2.25 \times 10^6 \) packets. We evaluate a cache in an access network having a few thousands of subscribers, and hence the total request rate of objects at the cache is 0.05 requests/s, where each subscriber downloads one object per day.

The traffic-mix scenario focuses on a situation where large and small objects are mixed in order to evaluate effects of object size on looped replacement. The following two types of objects are deployed: One is \( 10^6 \) web objects as small objects and the other is 200 video objects as large objects. Web and video objects consist of 5 and \( 10^3 \) packets, respectively. The total request rates of web and video objects are 50 and 0.05 requests/s, respectively, so that
the numbers of packets of small and large objects arriving at the cache are the same.

In both the scenario, the request rate of the objects follows a Zipf distribution with parameter $\alpha = 0.8$ [11] and the cache size is set to 1% of the number of entire packets.

4.2 Results

4.2.1 Looped Replacement Probability. First, we investigate the significance of looped replacement under the VoD scenario. Figure 3 shows the cache hit and the looped replacement probability of the top 500 objects requested frequently, which are referred to as popular objects hereafter. The x-axis indicates object IDs sorted in descending order of their request rates, $\lambda_i$.

This result reveals that looped replacement occurs for most of the popular objects except some of the most popular ones. The looped replacement probabilities of the 4th to 126th popular objects are higher than 0.1. The 16th popular object has the highest looped replacement probability, and its cache hit and looped replacement probabilities are 0.50 and 0.24, respectively.

4.2.2 Looped Replacement Probability and Object Size. Next, we evaluate how object size affects the looped replacement probability under the traffic-mix scenario. Figure 4 indicates the cache hit and the looped replacement probability of the top 200 popular web and video objects. The highest looped replacement probability of the video objects is 0.247 while that of the web objects is 0.8. This reveals looped replacement is more likely to occur when the object size, i.e., the number of packets constitute an object, is large.

4.2.3 Causes of Looped Replacement. In order to identify causes of looped replacement, we focus on four factors: the looped replacement probability ($\tau^i$), the object size ($m_i$), the length of time being in state $3 (r^i_3)$, and the total number of interrupting packets ($a_i$).

The following relations may be intuitively satisfied according to the analytical model: increase in $r^i_3$ may cause increase in $\tau^i$ (Eq. (1)), increase in $a_i$ may cause increase in $r^i_3$ (Eq. (6)), and increase in $m_i$ may cause increase in $a_i$ (Eqs. (2) and (3)). To evaluate the effects of $m_i$ on the other three values, $\tau^i$, $r^i_3$, and $a_i$, we add one object, referred to as a target object, to the evaluation environment described in Section 4.1 and vary the size of the target object from $2.25 \times 10^4$ to $2.25 \times 10^6$ packets, i.e., 1% to 100% of the size of other objects. The request rate of the target object is set to that of 16th object, which has the highest looped replacement probability.

Figure 5 summarizes the results of the three values ($\tau^i$, $r^i_3$, and $a_i$). The x-axes of the three graphs indicate the size of the target object $m_i$. As expected, the number of interrupting packets $a_i$ increases as $m_i$ increases. This extends the time of the object being evicted from the cache $r^i_3$, and hence looped replacement is more likely to occur.

Since the object size is a given value and it is uncontrollable with caching algorithms, we focus on the number of interrupting packets to suppress looped replacement. Figure 6 shows the proportion of the cumulative number of interrupting packets to the total number of interrupting packets. The x-axis indicates object IDs sorted in descending order of their request rates $\lambda_i$. The cumulative number of interrupting packets is defined as the sum of the number of interrupting packets of the top $k$ popular objects and it is derived as $\sum_{j=0}^{k} a_{ij}$, where object $i$ is the target object.

The proportion of the number of interrupting packets of the top 500 objects accounts for only 35.1% of the total number of interrupting packets $a_i$. In other words, 97% of the unpopular objects accounts for 64.9% of $a_i$. The analyses imply that to admit popular objects into the cache is a solution to suppressing looped replacement since it reduces $a_i$.

5 RESOLVING LOOPED REPLACEMENT

5.1 Cache Eviction and Admission

The goal of this section is to confirm not to admit packets requested infrequently, referred to as unpopular packets, to the cache suppresses looped replacement. Cache eviction algorithms determine a victim, a packet evicted from a cache, whereas cache admission algorithms decide whether an arriving packet is inserted to the cache or not. A cache admission algorithm is generally used in conjunction with a simple cache eviction algorithm like LRU and FIFO.

We use TinyLFU [10] and Filter [18] as cache admission algorithms. Both of the algorithms sort the number of recent arrivals of packets as the frequency of the packets and determine cache admission according to the frequency. They are, however, different in terms of their cache admission decision and their frequency management. Regarding the cache admission decision, TinyLFU admits an incoming packet if its frequency is higher than that of a victim packet chosen by a cache eviction algorithm but Filter does if its frequency is higher than the predefined threshold $\theta$. Regarding the frequency management, TinyLFU halves all the frequency in a predefined interval $W$ but Filter counts the frequency of only the past $Q$ arrivals. Both manage the frequency with simple data structures, and thus their time complexity is constant regardless of the cache size and the number of packets.
(b) The length of time in state 3 (\(\tau_i\))

(c) The number of interrupting packets (\(\alpha_i\))

Figure 5: Effects of the size of the target object on the looped replacement probability, the length of time in state 3, and the number of interrupting packets

Figure 6: The proportion of the cumulative number of interrupting packets

5.2 Evaluation Condition

It consumes a huge amount computing resources to simulate packet-level caching unlike the analyses in Section 4. We therefore evaluate it under a situation where a relatively small number of objects and packets are deployed due to limitations on our computing resources. 2,000 equal-sized objects, of which size is \(1.5 \times 10^5\) packets, are deployed. We use the same condition as Section 4 regarding the request rate of each object. The cache size is set to 1% of the number of all the packets.

Cache hit probability and looped replacement probability are measured as follows. The number of cache hits of an object is counted by the number of cache hits of the first packet of the object. Since packets of an object are requested sequentially from its first to its last packets, if a cache hit of the first packet hits occurs, cache hits of all the succeeding packets of the object are highly likely to occur. In the same way, the number of occurrences of looped replacement of an object is counted by the number of cache misses of the first packet of the object in the case that packets of the same object exist in the cache.

We use the LRU algorithm as a cache eviction algorithm in conjunction with the cache admission algorithms. For comparison, we also evaluate looped replacement probability of OPC [19], which uses objects as its unit of cache management and evicts packets of an object in the order of its last to its first packets.

5.3 Results

5.3.1 Suppression of Looped Replacement. Figure 7 shows the cache hit and the looped replacement probability of the top 100 popular objects. The graph indicates the mean and the 95% confidence intervals of results of 10 simulation trials. Let us note that the confidence intervals are sufficiently small, and error-bars are hidden behind markers. The x-axis indicates object IDs in the same way as Fig. 3. Filter has two parameters, the history size \(Q\) and the threshold \(\theta\), and TinyLFU has one parameter \(W\) for refreshing the frequency. We set \(Q\) to \(2C\), where \(C\) is the cache size, and \(\theta\) to 2, and \(W\) to \(128C\), which achieve the lowest looped replacement probability in the evaluation later.

Filter, TinyLFU, and OPC realizes the higher cache hit and the lower looped replacement probabilities than those of LRU. Trends of the improvements are, however, slightly different in the case of the admission algorithms and OPC. The admission algorithms improve the cache hit probabilities of popular objects but decrease those of unpopular objects since they do not insert unpopular objects into the cache. In contrast, OPC improves the cache hit probabilities of all objects. Let us note that comparing the absolute values of cache hit probability of the cache admission algorithms and OPC is meaningless in this evaluation. The two cache admission algorithms admit packets based on the frequency of the packets even though they evict packets based on recency of the packets. In contrast,
OPC evicts packets only based on the recency of the packets. This means that TinyLFU and Filter are good at handling requests of which rate do not change over time. Evaluating the algorithms with time-dependent requests is still subject to further study. OPC obviously suppresses looped replacement, and the looped replacement probabilities of all objects are 0. Filter and TinyLFU also suppress looped replacement mostly, and the maximum looped replacement probabilities of Filter and TinyLFU are 0.016 and 0.033.

5.3.2 Number of Interrupting Packets. To show that the cache admission algorithms reduce the number of interrupting packets, we measure the proportion of the cumulative number of interrupting packets to the total number of interrupting packets in the same way as Fig. 6. The result of the 2nd popular object, which has the highest looped replacement probability in the case of LRU, is shown in Fig. 8. Let us note that both the cache admission algorithms reduce the total number of the interrupting packets by approximately 70% compared to LRU. The proportions of the number of interrupting packets of the top 100 objects in the case of Filter and TinyLFU account for 94.9%, and 99.6%, respectively. These results indicate that cache admission does not admit unpopular packets and reduces the number of interrupting packets caused by them, thereby suppressing looped replacement.

5.3.3 Filter vs. TinyLFU. To evaluate which cache admission algorithm is suitable to suppress looped replacement, we evaluate them by changing the parameter $Q$ of Filter to 2C and 64C and the parameter $W$ of TinyLFU to 16C and 128C.

Filter reduces the looped replacement probability in the case of small $Q$ and TinyLFU similarly does in the case of large $W$, as shown in Fig. 9. With small $Q$ and large $W$, both Filter and TinyLFU strictly admit packets. More precisely, Filter remembers packets arriving recently in its queue, of which size is $Q$, and counts the frequency of packets in the queue. With small $Q$, only highly popular packets are likely to occupy the queue, and hence only the highly popular packets are admitted. In the case of TinyLFU, the frequencies of packets are halved in an interval $W$. Since we simulate packet arrivals according to the independent reference model with the constant arrival rate, TinyLFU is able to count the precise frequency of packets in the case of large $W$. This enables TinyLFU to identify highly popular packets precisely, thereby admitting only the highly popular packets. Filter with small $Q$ and TinyLFU with large $W$ therefore reduce the number of interrupting packets, and hence the occurrence of looped replacement is suppressed accordingly.

Let us note that adapting to changes in packet arrival rate over time is one of the important goals of TinyLFU [10]. TinyLFU with large $W$ is not able to adapt to changes in packet arrival rate because it remembers the long-term frequency of packets. In contrast, Filter with small $Q$ remembers the short-term frequency of packets and hence might be able to adapt to changing arrival rates as well as it suppresses the occurrence of looped replacement. Finally, we propose to use Filter to suppress the occurrence of looped replacement.

6 RELATED WORK

We compare our study with two series of related work.

First, looped replacement is resolved by OPC [19], a hybrid of packet-level and object-level caching. OPC tries to resolve limited storage resources as well as looped replacement. The problem of limited storage resources is that data structures to manage packets in the cache cannot be placed on fast but small memory like SRAM because the number of packets is much larger than that of objects. Though OPC successfully resolve the issues of packet-level caching, including looped replacement, it may restrict application scopes as we have discussed in Section 1. In contrast, this study proposes a simple but efficient countermeasure against looped replacement that is applicable to ICN instances with any kinds of naming schemes.

Second, several analytical models for object-level caching have been developed. Che et al. have developed an analytical model of object-level caching in [8]. We design an analytical model of packet-level caching by extending their model with the following two aspects. One is that we incorporate the object size, i.e., the number of packets of objects, into the model and the other is that we model the time gap between arrivals of the first and the last packets of a same object because packets are inserted and evicted individually.

7 CONCLUSION

Due to the increase in the size of content, the problem of looped replacement, which is a phenomenon that packets of a newly arriving object at a cache result in cache misses even though a part of the object is in the cache, arises in the area of packet-level caching of ICN. This paper has analytically investigated how looped replacement occurs. Through the analyses, we have an implication that not to admit unpopular packets to the cache is one of solutions to looped replacement. Finally, through simulation, we have confirmed that it suppresses the occurrence of looped replacement to use cache admission algorithms, Filter and TinyLFU.
ACKNOWLEDGMENTS
This work has been supported by JSPS KAKENHI Grant Number JP17H01733.

REFERENCES


