Design and Evaluation of a Utility-based Caching Mechanism for Information-centric Networks

Aifang Xu Xiaodong Tan Ye Tian*
School of Computer Science and Technology
University of Science and Technology of China, Hefei, Anhui 230026, China
Email: {xaf, xdtan}@mail.ustc.edu.cn, yetian@ustc.edu.cn

***Abstract*—Information-centric networking (ICN) is one promising direction for the future Internet, and how to manage the in-network caching resources is a fundamental problem in ICN. In this paper, we address the problem by proposing a utility-based caching mechanism. In the mechanism, network nodes track the *utilities* of the contents that they have ever cached, and en-route nodes cooperate to make the caching decisions. For enabling the utility tracking, we introduce a novel component named *Tracking Store* in ICN routers, and develop two methodologies for implementing this component based on dynamic LRU queue and time-decaying Bloom filter (TBF). Through analysis and extensive simulations using real-world topologies, we show that at a sustainable router overhead, our proposed mechanism, with both of its implementations, achieves a superior caching performance than existing solutions under various content popularity scenarios. We also explore the inherent tradeoff of the mechanism, and provide guideline for its real-world deployment.**

I. Introduction

With information becoming the first-class citizen on the Internet, a number of information-centric network (ICN) architectures, such as DONA [1], CCN [2], and COMET [3], have been proposed in recent years. Using CCN [2] as example, we briefly describe the basic idea of ICN. In CCN, a content object is divided into equal-sized chunks, and each chunk has an URL-like name that is globally unique. When a router receives a chunk requesting message (named *Interest* in CCN), it looks up its *Forwarding Information Base* (*FIB*) using the chunk name, and forwards the request towards the content servers. After that, the request is kept in router's *Pending Interest Table* (*PIT*), until the corresponding content chunk traverses the node along the reverse path or timeout. However, when the router has already cached the chunk in its local *Content Store* (*CS*), it can directly respond the request with its local cached replica and drop the request.

Unlike the conventional forwarding-only network, caching is pervasively applied in ICN, where many content requests, especially the ones for the popular contents, can be answered by routers rather than the content servers. With such a pervasive in-network caching, how to manage the caching spaces on the network nodes becomes one of the fundamental problems in ICN study.

*Correponding author
This work was supported by the National Natural Science Foundation of China (No. 61202405) and the sub task of the Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDA06010302).

There exist two paradigms in the ICN cache management, namely the *en-route caching* (e.g., [2], [4], and [5]) and the *collaborative caching* (e.g., [6] and [7]). In en-route caching, only the nodes along the path between the content origin (i.e., content server or the router that answers the content request using its locally cached replica) and the content requesting client are considered as candidates for caching the replica. On the other hand, in collaborative caching, nodes from the entire network can be the candidates, and these nodes explicitly exchange messages to make the caching decisions in a collaborative way. Comparing with en-route caching, collaborative caching may lead to a higher cache hit ratio since more nodes are involved; however, the merit of en-route caching is its simplicity, which is very desirable for the practical deployment in high-speed networks.

The original CCN network proposal [2] adopts en-route caching and employs a "leave a copy everywhere" (*LCE*) strategy. In LCE, the network caches a replica of the content on every router along the content's traversing path. However, studies [4][5] shows that by caching duplicate replicas, LCE indeed wastes the precious storage space on routers and degrades the network's caching performance.

On the other hand, recent works propose to keep only one replica at a carefully selected position on the content traversing path, so as to avoid the redundancy and promote the usefulness of the cached replica. There are two representative approaches: one is to "keep the replica close to the content origin". For example, the "leave a copy down" (*LCD*) strategy [8] which is originally proposed for multi-level cache in computer architecture, has been considered to apply under the context of ICN recently [9]. In LCD, one replica is cached at the node that is immediately downstream from the node that answers the content request. The rationale behind LCD is that since content requests from all over the network will be multiplexed at the nodes that are close to the origin, then by caching contents at these positions, the network will be more likely to intercept these requests.

The other representative approach is to "keep the replica at important positions". For example, in the topology-aware caching strategy (referred to as *TA* for short) proposed in [4], a replica is cached at the node that has the largest betweenness centrality along the path. The TA strategy also has its rationale, that is, by keeping replicas at the important positions, a higher cache hit ratio could be expected.

In this paper, we focus on the en-route caching paradigm, and propose a utility-based caching mechanism for ICN networks. Unlike the existing solutions that are based on the metrics such as the distance from the content origin (as in LCD) and the betweenness centrality (as in TA), which only indirectly reflect the usefulness of the content replica, in our proposed mechanism, we allow a router to directly measure the *utility* of a content replica that it has ever cached, and after the replica has been evicted, the router will still track the content's utility for some time. When a new replica of the same content needs to be cached on the network, the content origin collects the utilities of the content that are tracked by the nodes along the path, and uses the information to make the caching decision. To fulfill the utility-based caching mechanism, we make the following efforts in this paper:

- We introduce a novel component named *Tracking Store* (or *TS* for short) in ICN routers for enabling the router to track the content utilities; we also extend ICN's basic content request and response messages to piggyback the utility information tracked by the en-route nodes, and assist the network to make the caching decision based on the utilities.
- We develop two methodologies based on dynamic LRU queue and time-decaying Bloom filter (TBF) to implement the TS component, and we analyze the overheads on routers that are incurred by our proposed caching mechanism.
- We extensively evaluate our proposal using real-world Internet topologies, and find that our proposed mechanism, with both of its implementations, outperforms its existing solution counterparts; we explore the mechanism's inherent tradeoff, and provide guideline for deploying it in real world.

The remainder part of this paper is organized as follows: In Section II, we propose the utility-based caching mechanism, present its implementing methodologies, and analyze its overheads; In Section III, we evaluate the performance of our proposal and compare it with existing solutions; We conclude this paper in Section IV.

II. Utility-based Caching Mechanism

A. The Basic Design

We first describe the basic idea of the utility-based en-route caching mechanism. We assume that our proposed mechanism is working on a typical ICN network as in [2], where each router employs a Content Store (CS) to cache the chunk replicas.

1) The Tracking Store component: In our proposed mechanism, for each chunk replica cached by an ICN router in the CS component, in addition to its name and the content data, the router also keeps a counter named hit to indicate how many times this replica has been hit since it was cached. For example, for a newly cached replica, its initial hit value is the number of the arrival interfaces of the corresponding content request in PIT minus one, and if the router uses its locally cached replica to respond a content request, the replica's associated hit value will be incremented by one.

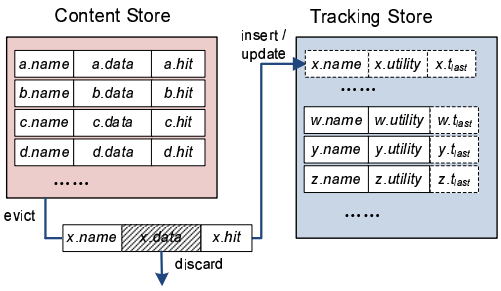


Fig. 1. The Tracking Store (TS) component

In addition to CS, in each ICN router we introduce a new component named *Tracking Store* (or *TS* for short). As demonstrated in Fig. 1, TS employs an LRU queue where each entry tracks the *utility* of a content that the router has ever cached and evicted from CS. More specifically, each entry in TS a <key: value> pair as $< name : (utility, t_{last}) >$, where $name$ is the name of the content under track, $utility$ is a non-negative value indicating how useful if the content is cached on this router, and t_{last} is the latest time that the entry was updated. Note that each entry in TS only keeps the metadata such as the name and utility value of the content without storing the actual content data, therefore a router would be able to track much more contents in TS than it is capable to cache in its CS component. A more detailed analysis on the storage overhead of the TS component can be found in Section II-B3.

As we can see in Fig. 1, when a replica of a content, say content x, is evicted from CS, the ICN router examines the replica's associated counter $x.hit$, if $x.hit > 0$, suggesting that this replica has been hit at least once, the router inserts or updates x's corresponding entry in TS as follows: the router first looks up TS using the content's name $x.name$, if there is no entry for it, the router directly uses $x.hit$ as x's utility, i.e., $x.utility = x.hit$, and inserts an entry like $< x.name : (x.utility, t_{cur}) >$ at the head of the LRU queue in TS, where t_{cur} is the router's current local time; On the other hand, if there already exists an entry as $< x.name : (x.utility, t_{last}) >$ in TS, the router updates x's utility using the following equation:

$$x.utility = \begin{cases} x.hit, & \text{If } x.utility \times \rho^{t_{cur}-t_{last}} < x.hit \\ \alpha \times x.hit + (1-\alpha) \times x.utility \times \rho^{t_{cur}-t_{last}}, & \text{Else} \end{cases} \quad (1)$$

where α $(0 < \alpha < 1)$ is the weight of the hit times of the newly evicted replica in utility, and ρ $(0 < \rho < 1)$ determines how utility decays over time. Equation (1) suggests that x's utility value is indeed an exponential weighted moving average of the hit times of all the content x replicas that have ever been cached by the router; however, if the updated utility is smaller than the hit times of the newly evicted replica, i.e., $x.hit$, the router directly uses $x.hit$ as the utility value without calculating the moving average.

Algorithm 1 Utility update algorithm

```
1: procedure UPDATE_UTILITY(x)
2:                                  ▷ x is a newly evicted replica from CS
3:     if x.hit > 0 then
4:         if Entry of x.name exists then
5:             Update x's utility using Equation (1);
6:             Move the updated entry to the TS queue head;
7:         else
8:             x.utility ← x.hit;
9:             Insert a new entry < x.name : (x.utility, t_cur) > at
   the TS queue head;
10:        end if
11:    end if
12: end procedure
```

After computing the utility value, the router removes the old entry and places a new entry with the updated utility value as $< x.name : (x.utility, t_{cur}) >$ at the head of the LRU queue in TS. Algorithm 1 presents a formal description of the utility update algorithm.

2) Making caching decisions: With each ICN router equipped with a TS component, which enables it to track utilities of the contents it has ever cached, we show how the network can employ such a tracking capability to make the caching decisions. Consider an example in Fig. 2, where a client connecting to node v_1 sends out a request for content x, the requesting message is forwarded along the path towards the content origin (i.e., the content server s). For each router on the path, in addition to forward the request message, the router also looks up its TS using $x.name$, and if there exists an entry $< x.name : (x.utility, t_{last}) >$, the router decays x's utility value in the entry with current time t_{cur} as $x.utility \times \rho^{(t_{cur}-t_{last})}$, and attaches the decayed utility value together with the router ID to the message, as we can see in Fig. 2; In case that no entry for content x is found, the router simply forwards the request without attaching any utility information.

After the content origin s receives the content request, it also receives with the message an array of the utilities from all the routers along the path. The content origin makes the caching decision based on the utilities. In this paper, we allow the content origin to select the router with the highest utility value, for example, node v_{k-1} in Fig. 2, to cache the replica. The caching decision can be piggybacked to the chunk that is sent towards the requesting client along the reverse path.

B. Implementation

1) Determining dynamic LRU queue size: In the above description of the utility-based caching mechanism, we employ an LRU queue to implement the TS component. It is very necessary to determine an appropriate size of the queue: If the queue is too small, the network may not be able to make proper caching decisions, as only limited number of the contents can be tracked; on the other hand, if the queue is too large, precious storage spaces on ICN routers are wasted. Following we present a methodology for dynamically determining the LRU queue size in the TS component.

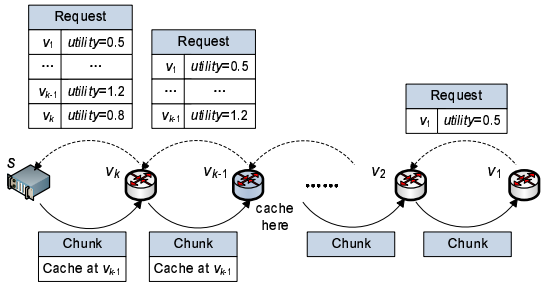


Fig. 2. A example of caching decision made based on content utilities that are piggybacked on ICN's chunk request and response messages

For each ICN router, it keeps a global variable named $u_{\max}$, which records the largest utility value the router has observed during a recent period of time. We also define a threshold $u_{\min}$ for the "useless" content, that is, for any content tracked by a router, if its utility becomes less than $u_{\min}$, we consider it as useless and no longer needs to be tracked. For the LRU queue in TS, when an entry with a utility value of $u_{\max}$ is placed at the head of the queue, according to Equation (1), the time it takes for $u_{\max}$ to be decayed to as small as $u_{\min}$ is

$$t_{\max} = \log_\rho u_{\min} - \log_\rho u_{\max}$$

where ρ is the decaying factor.

Suppose that each time an entry is placed at the head of the LRU queue, we consider it as a "new" entry, and we suppose that an entry will not be evicted from TS until it becomes useless, then $t_{\max}$ can be considered as an optimistic estimation of the lifetime that an entry can stay in TS. Let λ_{TS} be the rate that entries are placed at the LRU queue head, regardless whether they are updated from existing entries or newly inserted, then by Little's law, an optimistic estimation of the LRU queue size is

$$L = \lambda_{TS} \times t_{\max} = \lambda_{TS} \times (\log_\rho u_{\min} - \log_\rho u_{\max}) \quad (2)$$

We dynamically adjust a node's TS queue size as the following: each ICN router monitors the rate λ_{TS} that entries are placed at the LRU queue head, and it also records the maximum utility $u_{\max}$ it has observed in a recent time window; periodically, the node applies Equation (2) to adjust its LRU queue size.

From Equation (2) we can see that the smaller the decaying factor ρ is, the shorter the LRU queue will be. However, there is a price, as we will see in Section III-D, a smaller ρ value will lead to a lower cache hit ratio of the network.

2) A Time-decaying Bloom filter implementation: For further reducing the storage usage when tracking a large number of contents, we propose a methodology based on the time-decaying Bloom filter (TBF) [10][11] to implement the TS component. As demonstrated in Fig. 3, the TBF-based TS is implemented as an array $b[1 \cdots m]$ of m cells, where each cell contains a non-negative value representing the utility of the content that is mapped to the cell.

The mapping between the content names and the TBF cells are determined by k hash functions: H_1, H_2, ..., H_k. More

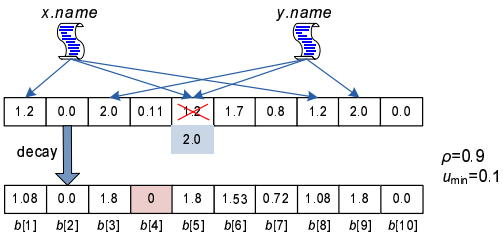


Fig. 3. Demonstration of a time-decaying Bloom filter for TS implementation

specifically, when a content x's utility is inserted in TS, for all the cells with their indices as $H_1(x.name)$, $H_2(x.name)$, ..., $H_k(x.name)$, their values will be set as the content's utility, that is, for $i = 1, \ldots, k$, we have $b[H_i(x.name)] = x.utility$. Moreover, a cell's value will be overwritten by a new content if the content has a different utility value and its name is hashed to the same cell. For example in Fig. 3, $b[5]$ is first set by content x with its utility value 1.2, but when the Bloom filter is updated by another content y, $b[5]$ is changed to 2.0, which is y's utility value.

In our TBF-based TS, cells decay over time. After each time interval, a cell's value is decayed with the decaying factor ρ ($0 < \rho < 1$), that is, for $\forall j$, $b[j]$ is decayed to as $b[j] \leftarrow \rho \times b[j]$. Moreover, when a cell's value becomes less that $u_{\min}$, which is the "useless" threshold, it is cleared to zero. For example in Fig. 3, after the decaying, cell $b[4]$ is cleared. Finally, to query a content's utility in the TBF-based TS, we use the minimum value of all the cells that the content is mapped to. That is, for content x, the TBF-based TS returns its utility value as $x.utility = \min_{i=1,\cdots,k}(b[H_i(x.name)])$.

There are two errors that could be introduced by TBF. One is the Bloom filter's inherent false positive error, that is, the TBF may return a non-zero utility for a content that is not tracked by TS. The other error is that the queried utility value may be smaller than the real one, as we use the minimum value among all the cells to return the query. We will discuss the influence of the TBF errors in Section III-E.

3) Overhead discussion: We analyze the storage and communication overheads that are incurred by our proposed caching mechanism on routers.

Storage overhead We first look at the TS implementation based on the dynamic LRU queue. According to [12], for the LRU-based TS, its hash table would consume $136n$ bits, where n is the number of the entries in TS, and if we use 16 bits to represent a utility value, and 32 bits for storing the latest update time, then the total memory space required is $Mem_{LRU-TS} = (136 + 16 + 32)n = 184n$ bits. Considering a typical chunk size of $10K$ bytes [9], we can see that the storage overhead for tracking a content is $\frac{184}{8\times 10K} \approx 0.22\%$ of the cost for caching it; and if we implement the TS component on a 16M SRAM chip, a single chip is capable to track the content chunks of a total size of 6.96 gigabytes.

We then show that the storage overhead can be further reduced with a TBF-based implementation. Note that in TBF, the memory required for tracking n contents is $Mem_{TBF-TS} = -16\frac{n \ln p_{fp}}{(\ln 2)^2} \approx -33.3(\ln p_{fp})n$ bits, where p_{fp} is the Bloom filter's false positive probability [10]. Comparing with Mem_{LRU-TS}, we can see that, as long as p_{fp} is larger than 0.4%, the TBF-based TS implementation would be more storage-efficient with $Mem_{TBF-TS} \leq Mem_{LRU-TS}$.

Communication overhead By piggybacking on ICN's content request and response messages, our proposed caching mechanism incurs additional communication overheads. For example, if we use the MAC address as the router ID and use 16 bits for a utility value, then each router will attach at most 54 bits on the content request message. We believe the overhead is not significant for two reasons: First, it is reported that the Internet's averaged path length in terms of ASes is less than four and exhibit a decreasing trend [13]; Second, with the pervasively deployed in-network caching capacity, many contents can be served by routers, thus the distance between the content requester and the content provider will be significantly reduced on ICN.

III. Performance Evaluation

In this section, we evaluate the utility-based caching mechanism with simulation-based experiments. We employ real-world topologies of large ISPs in our evaluation, and compare our proposal with existing solutions of LCE [2], LCD [8], and TA [4] as introduced in Section I.

A. Simulation Setup

In our simulation, we use the PoP-level topologies of two large ISPs: one is Cogent in Europe and North America, which we download its topology containing 180 nodes and 212 links from Topology Zoo [14]; the other ISP is ChinaNet, which has the largest commercial backbone network in China, and we obtain its topology containing 278 nodes and 908 links from [15]. Note that the two ISPs represent two types of Internet topologies. The ChinaNet topology is a typical small-world network with a heavy-tailed node degree distribution, and there exist a few "hub nodes" connecting many neighbors in the network; on the other hand, the Cogent network is much more regular than ChinaNet, exhibiting a less skewed node degree distribution.

For each emulated network in our simulation, we randomly select four nodes to connect to the content server clusters, thereby servers at each node serve one fourth of the entire content catalog. Each node in the network is connected by a large number of clients that request and consume contents all the time. We assume that averagely the clients at each node request five content chunks per second with a Poisson process.

We consider a catalog of as many as 500 million distinct content chunks on the ICN network, and assume that the content popularity follows a Mandelbrot-Zipf (MZipf) distribution [16], where the probability of the k^{th} most popular content being requested is

$$p_k = \frac{1/(k+q)^s}{\sum_i 1/(i+q)^s} \tag{3}$$

here s is the model's skewness factor and q is referred to as the plateau factor. We denote the popularity model as

$MZipf(s,q)$. To name the contents, we employ the 100 million most popular domain names obtained from Alexa, and create 500 million distinct names from them[1].

For an ICN network, most of its advantages are based on its in-network caching capacity, we therefore focus on the cache hit ratio in our evaluation. More specifically, for a caching mechanism under study, we consider its *performance gain*, which is defined as the cache hit ratio accomplished by the mechanism divided by the cache hit ratio achieved by the LCE strategy, as LCE is the default solution in the original CCN proposal [2]. We assume routers apply LRU as the eviction policy in CS. For the parameters used in the utility-based caching mechanism, we let $\rho = 0.95$, $\alpha = 0.6$, $u_{min} = 0.1$, and $p_{fp} = 0.01$ if not otherwise specified.

B. Impact of Content Popularity Model

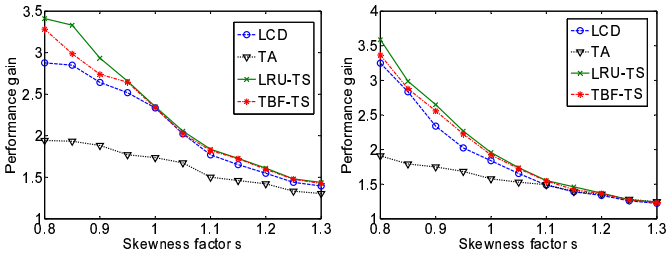
In our first experiment, we evaluate the performance of our proposed utility-based caching mechanism, engaging both the LRU-based and the TBF-based implementations (referred to as *LRU-TS* and *TBF-TS* respectively), and compare with the existing strategies of LCD [8] and the topology-aware (TA) approach [4]. We explore a wide range of popularity models by varying the MZip distribution's skewness parameter s from 0.8 to 1.3, and carry out the simulation on both Cogent and ChinaNet topologies.

In Fig. 4, we present performance gains of the different ICN caching solutions. From the figures we can make the following observations: First, all the approaches under evaluation, namely LRU-TS, TBF-TS, LCD, and TA, significantly outperform LCE with their performance gains larger than one, which conforms to the previous study [4] that better performance can be achieved by avoiding caching duplicated replicas. Second, LRU-TS and TBF-TS have higher performance gains than LCD and TA under all the popularity models, and LCD outperforms TA in most cases; in fact, only under the small-world ChinaNet topology, TA starts to outperform LCD when the contents' popularity is skewed enough ($s > 1.2$); however, under this case, there is actually little differences among the four schemes, as all of them can easily identify the popular contents. Finally, we find that LRU-TS is slightly better than TBF-TS under all the circumstances, which can be explained with the errors that are introduced by the Bloom filter.

C. Impact of Content Popularity Dynamics

In the previous experiment, we assume that a content's popularity is static, that is, when the content is assigned with a popularity, it will never change. However, in real world, a content's popularity will change over time. In this experiment, we consider a dynamic popularity scenario. More specifically, for each content, say the k^{th} content, we define its *accumulative popularity* as $A_k = \sum_{i=1}^{k} p_i$, where p_i is the content's popularity defined in Equation (3). In our experiment, in every 60 seconds, we randomly select a number of content pairs,

[1]For example, for the name "google.com", we create five names as "1.google.com", ..., "5.google.com".



(a) Cogent (b) ChinaNet

Fig. 4. Performance comparison of different caching solutions under the Cogent and ChineNet topologies

where in each pair, one content has its accumulative popularity in the range of $(0, 0.5]$ and the other has its accumulative popularity in $(0.5, 1]$, and we switch the popularities of the two contents.

In Fig. 5, we present performances of different en-route caching solutions of LCD, TA, LRU-TS and TBF-TS under the popularity dynamics. For each plot on the figure, its x-axis is the ratio of the contents with their popularities changed in the entire content catalog, and the y-axis is the performance gain. From the figure one can see that the dynamics of the content popularity can considerably influence the network's performance, as the performance gains of all the caching schemes degrade when the dynamic ratio becomes larger. In addition, we find that our proposed utility-based caching schemes of LRU-TS and TBF-TS can better handle the popularity dynamics, while TA has the worst performance. The better performances of our solutions can be explained with Equation (1), in which a content's utility decays over time, but can be updated by a newly evicted replica, so as to be adaptive to the content's recent popularity.

D. Evaluating Dynamic Queue Size in LRU-TS

In Section II-B, we present a methodology for dynamically determining the LRU queue size for the LRU-based TS implementation. In this experiment, we examine the effectiveness of this methodology, and compare it with the approach that each router in the ICN network maintains a fixed-sized LRU queue in its TS component.

In Fig. 6, we plot the performances of the utility-based caching mechanisms using two different decaying factor ρ values, and we employ both the dynamic-sized LRU queue and the fixed-sized queue in the LRU-based TS implementation under different ρ values. Note that for the dynamic-sized LRU queue, we have only one plot on the figure, whose x-axis is the averaged queue size as determined by Equation (2) for all the nodes within the network; while for the fixed-sized approach, we vary the queue size from 50 to 400 entries, and plot the performance gains. From the figure, we can see that under the approach with the fixed-sized LRU queue, when the queue is small, increasing its size can improve the network's performance, but when the queue is large enough, further expanding it barely helps. Moreover, for our proposed methodology with the dynamic-sized LRU queue, the network

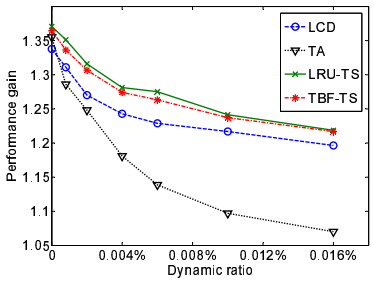


Fig. 5. Performance comparison of different caching solutions under popularity dynamics, employing the ChinaNet topology and the $MZipf(1.2, 10)$ popularity model

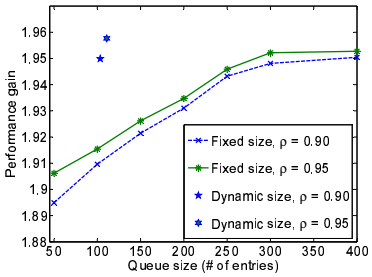


Fig. 6. Comparison between the dynamic-sized queue and the fixed-sized queue in the LRU-based TS implementation, under the ChinaNet topology and the $MZipf(1.0, 10)$ popularity model

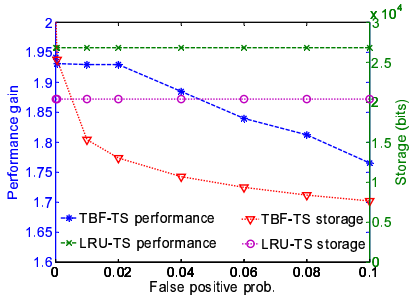


Fig. 7. Comparison between the TBF- and LRU-based TS implementations under various false positive probabilities, employing the ChinaNet topology and the $MZipf(1.0, 10)$ popularity model

achieves higher performance gains, but employing much less entries on the ICN nodes on average.

From Fig. 6, one may find that a larger decaying factor ρ leads to a better performance. However, there are pros and cons: under a dynamic popularity scenario as we have discussed in Section III-C, a large decaying factor will make the caching mechanism less responsive to the popularity dynamics, and it will also cause more storage usages as we can see from Equation (2).

E. Impact of Bloom Filter Errors in TBF-TS

In our last experiment, we focus on the TBF-based implementation, and explore how the errors that are introduced by the Bloom filter can influence an ICN network's caching performance. In our experiment, we vary the expected false positive probability p_{fp} from 0.001 to 0.1 when implementing the TBF-based TS component, and examine the network's performance gain and the memory usage of the TS component on each node. Fig. 7 shows our experimental results. We also plot the performance gains and the storage usages under the LRU-based TS implementation for comparison.

From Fig. 7, we can see that: when the false positive probability is very small, the TBF-based TS implementation has a performance gain that is very close to the one achieved by the LRU-based implementation, but employs even more storage. When p_{fp} becomes larger, the TBF-based TS implementation consumes less storage, but at cost of a lower performance gain. From the figure we can see that in real-world deployment, the false positive probability p_{fp} can serve as a tuning parameter, which enables the network administrator to trade an ICN network's caching performance with the storage usage on routers in the ICN network.

IV. Conclusion

In this paper, we address the fundamental problem of the in-network caching management in ICN networks. We propose a utility-based en-route caching mechanism, and present two implementing methodologies which are based on dynamic LRU queue and time-decaying Bloom filter (TBF). From simulation experiments using real-world topologies, we show that our proposed mechanism outperforms existing solutions with higher cache hit ratios; moreover, we find that the proposed implementing methodologies are storage-efficient and provide flexibilities in real-world deployment.

References

[1] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proc. of ACM SIGCOMM'07*, Kyoto, Japan, Aug. 2007.

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. of CoNEXT'09*, Rome, Italy, Dec. 2009.

[3] G. Garcia, A. Beben *et al.*, "COMET: Content mediator architecture for content-aware networks," in *Proc. of Future Network & Mobile Summit*, Warsaw, Poland, Jun. 2011.

[4] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache 'less for more' in information-centric networks," in *Proc. of IFIP Networking'12*, Seattle, WA, USA, May 2012.

[5] I. Psaras, W. K. Chai, and G. Pavlou, "In-network cache management and resource allocation for information-centric networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 2920 – 2931, 2014.

[6] S. Guo, H. Xie, and G. Shi, "Collaborative forwarding and caching in content centric networks," in *Proc. of IFIP Networking'12*, Prague, Czech, May 2012.

[7] L. Saino, I. Psaras, and G. Pavlou, "Hash-routing schemes for information centric networking," in *Proc. of SIGCOMM ICN Workshop (ICN'13)*, Hong Kong, China, Aug. 2013.

[8] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD interconnection of LRU caches and its analysis," *Performance Evaluation*, vol. 63, no. 7, pp. 609 – 634, 2006.

[9] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," Telecom ParisTech, Tech. Rep., 2011.

[10] K. Cheng, L. Xiang, and M. Iwaihara, "Time-decaying bloom filters for data streams with skewed distributions," in *Proc. Int. Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications*, Tokyo, Japan, Apr. 2005.

[11] G. Bianchi, N. d'Heureuse, and S. Niccolini, "On-demand time-decaying bloom filters for telemarketer detection," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 5, pp. 5 – 12, 2011.

[12] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proc. of SIGCOMM ICN Workshop (ICN'11)*, Toronto, Canada, Aug. 2011.

[13] B. Edwards, S. A. Hofmeyr, G. Stelle, and S. Forrest, "Internet topology over time," *CoRR*, vol. abs/1202.3993, 2012. [Online]. Available: http://arxiv.org/abs/1202.3993

[14] "The Internet Topology Zoo," http://www.topology-zoo.org.

[15] Y. Tian, D. Ratan, Y. Liu, and K. W. Ross, "Topology mapping and geolocating for china's internet," *IEEE Trans. on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1908 – 1917, 2013.

[16] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *IEEE/ACM Trans. on Networking*, vol. 16, no. 6, pp. 1447 – 1460, 2008.