Caching Policies for Transient Data

Santosh Fatale, Sri Prakash and Sharayu Moharir
Department of Electrical Engineering
Indian Institute of Technology Bombay
Email: sfatale@ee.iitb.ac.in, sriprakash@ee.iitb.ac.in, sharayum@ee.iitb.ac.in

***Abstract*—This work focuses on designing caching policies for transient data, i.e., data which can be used to serve requests only for a finite duration of time after which it becomes redundant. We first characterize the fundamental limit on the performance of caching policies for transient data and characterize the performance of traditional caching policies like LRU for this setting. Traditional caching policies often make decisions based on the popularity of the data being cached. We propose a new caching policy which uses both the popularity and the residual lifetime (time remaining before the data becomes redundant) to make caching decisions. We show that in the setting where data being cached is transient, our policy outperforms traditional caching policies.**

I. Introduction

The motivation for this work comes from the increasing popularity of the Internet of Things (IoT) and the concept of Information Centric Networking (ICN). ICN assigns a unique name to each piece of data and consumers request for a specific piece of data instead of requesting to communicate with the producer of the requested data. In addition, nodes in the network are equipped with storage capabilities and data can be cached at intermediate nodes in the network. Since ICN allows data to be cached close to the consumers, the benefits of ICN over IP-based communication include a reduction in retrieval delay, lower bandwidth consumption, and reduced energy consumption [1].

Since many IoT applications are information-centric, the idea of using ICN for IoT has gained traction [1]–[3]. The benefits of using ICN for IoT network have been explored in [2] where it was found that in-networking caching leads to lower delays and lower power consumption. In [3], it was observed that the performance of ICN for IoT depends heavily on the caching policies used by various nodes in the network.

In this work, we study systems consisting of multiple sensors called producers, each of which measures a time-varying signal, consumers, and an intermediate router as illustrated in Figure 1. Consumers interested in the measurements of these sensors access them via the intermediate router which is equipped with storage capabilities. In most IoT applications, the measurements of the sensors are used by the consumers to make control decisions. Since using stale measurements can lead to sub-optimal decisions, we focus on the setting where each request has a specific freshness requirement, i.e., a request received at time t for Producer i's measurement can only be served using a measurement collected after time $t - F_i$, where F_i is the freshness requirement of the request. The need for fresh measurements makes the data being cached transient, i.e., once fetched and stored, the data can be used to serve requests for a specific amount of time and becomes stale thereafter. The transient nature of data is the key difference between caching for IoT applications as compared to applications like Video-on-Demand.

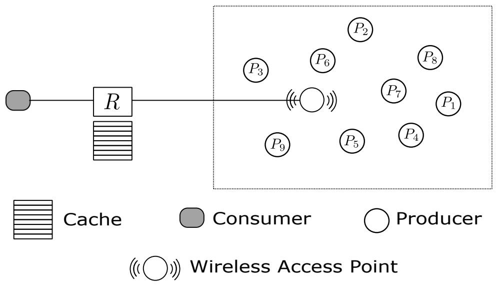

Fig. 1: A system consisting of multiple sensors, a router and consumers. The sensors measure time-varying signals and consumers interested in the measurements of the sensors access them via a router. The router is equipped with storage resources to cache data.

Traditional caching policies like Least Recent Used (LRU) and Least Frequently Used (LFU) make caching decisions based on the popularity of the data being cached. In this work, a consequence of the transient nature of the data being cached is that every piece of data is characterized by two attributes, namely, its popularity and its residual life-span, i.e., the time remaining before the data becomes stale and can't be used to serve requests. Since traditional caching policies make caching decisions purely based on the popularity of the data, they are sub-optimal to cache transient data. The goal of this work is to design efficient caching strategies for transient data.

A. Contributions

The main contributions of this work are as follows.

- We prove a fundamental limit on the performance of any caching policy (Proposition 1) and characterize the performance of traditional caching policies for caching transient data (Propositions 2-4). We validate our theoretical results via simulations.
- Next, we propose a new caching policy called Least Useful (LU) which uses the popularity as well as the residual life-span to make caching decisions. We provide a theoretical characterization of the performance of the LU policy (Proposition 5).
- We show that our policy outperforms traditional caching policies via extensive simulations.

B. Related Work

Caching policies for transient data have been studied in [4]–[10]. Closest to this work, in [4], the performance of a system similar is studied for a time-slotted setting where exactly one request arrives at the beginning of each time-slot. We focus on the setting where requests arriving according to a Poisson process. In [5], [6], the authors consider the setting where there are no hard freshness constraints, however, the system pays a cost if requests are served using stale data. In this work, requests have hard freshness requirements, thus differentiating our work from [5], [6]. In [7], [8], a single producer system is considered where the producer decides when to send updates to the cache (push-based communication). The updates sent by the producer enter a queue and are sent to the router according to queue's service process. In [7] the focus is on studying the impact of various queuing disciplines like M/M/1, M/D/1, D/M/1. In [8], the authors evaluate the benefits of employing packet management on the system studied in [7]. Like [7] and [8], [9] focuses on a push-based setting where sensors send updates to the cache. The caching problem is modeled as an optimization problem and the objective is to minimize the time average of the weighted sum of ages of the cached data. [10] focuses on caching strategies for the setting where the popularity of a content varies with its freshness. The key takeaway in [10] is that, in the setting where popularity varies with freshness, the optimal caching policy is to cache the most popular contents at a given time. Our work differs from [10] because, in our setting, the popularity of producers remains constant, however, a measurement taken from a sensor is useful only for a finite amount of time after it is taken.

II. Setting

We study a system consisting of N sensors (Figure 1), each measuring a different time-varying signal. We refer to these sensors as producers (since they are the source of data in the system). These producers communicate with a router equipped with limited storage capabilities. All entities interested in the producers' measurements can access them via this router.

A. Request Model

Requests for the producers' measurements arrive according to a Poisson process with rate one. The probability of an incoming request being for Producer i is denoted by λ_i. In addition, each request for Producer i's data has a freshness requirement F_i, i.e., a request made for Producer i's data at time t can only be served by a measurement made by Producer i after time $t - F_i$. The F_is are fixed and determined based on the nature of the signal each producer is measuring. For example, F_i is low if Producer i is measuring a highly time-varying signal and high for slowly varying signals. Examples of such processes include:

Example 1: (*Multi-class model*) The producers are divided into K classes where all producers in a class have the same λ value and freshness requirement. More specifically, the probability of an incoming request being for Producer i in Class k is $\lambda_i = \lambda_{(k)}$, and all requests for the data of Producer i in Class k have a freshness requirement of $F_i = F_{(k)}$. If the number of producers in Class k is denoted by N_k, the following condition is satisfied by the $\lambda_{(k)}$s:

$$\sum_{k=1}^{K} N_k \lambda_{(k)} = 1.$$

Example 2: (*Zipf popularity and uniform freshness*) The λ_is follow the Zipf distribution, i.e.,

$$\lambda_i = c(\beta) i^{-\beta}, \text{ with } c(\beta) = \left(\sum_{i=1}^{N} \lambda_i \right)^{-1},$$

where $\beta > 0$ is the Zipf parameter. Requests for all producers have a freshness requirement of F units.

B. Storage Model

The router has a storage capacity of C units where each producer's data occupies one unit of space. This one unit includes the value measured by the producer as well as the time-stamp indicating the time at which this measurement was fetched and cached.

C. Service Model

When a request for Producer i's data is received by the router (say at time t), it checks if it has a fresh measurement from Producer i stored (collected after time $t - F_i$). If found, the stored measurement is used to serve the request, else, the current value measured by Producer i is fetched by the router to serve the request. This newly fetched data can be stored by the router to serve future requests. Note that this work is restricted to the setting where the router does not fetch a new measurement from a producer unless it is required to do so to serve a request.

D. Goal

We refer to the event of an incoming request being served by a measurement stored at the router as a *hit* and the complimentary event as a *miss*. There are two types of misses: *(i)* the requested producer's data is not stored in the cache, and *(ii)* the requested producer's data stored in the cache does not satisfy the corresponding freshness requirement.

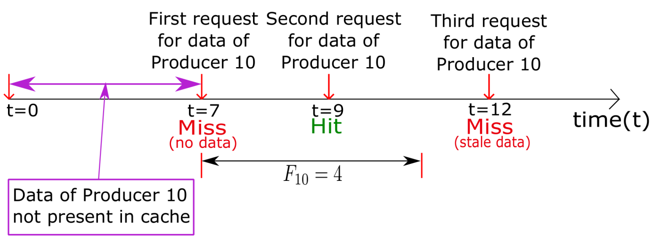
First request for data of Producer 10
Second request for data of Producer 10
Third request for data of Producer 10
t=0
t=7
Miss
(no data)
t=9
Hit
t=12
Miss
(stale data)
time(t)
$F_{10} = 4$
Data of Producer 10 not present in cache

Fig. 2: An illustration of a hit and the two types of misses.

For example (Figure 2), consider the case where Producer 10 has a freshness requirement of 4 units. For a system starting at $t = 0$, the first request for Producer 10 is received at $t = 7$. Since this is the first request for Producer 10, this leads to a miss of the first type, i.e., the data of the requested producer is not present in the cache. Assume that the data fetched from Producer 10 at $t = 7$ is cached and not evicted for the next 5 time-units. When the next request is received at $t = 9$, the stored data satisfies the freshness requirement and therefore, the request leads to a hit. The third request, received at $t = 12$ leads to a miss because the stored data is 5 units old and therefore, leads to a miss of the second type.

The goal is to design a caching policy which determines which producers' measurements should be stored at the router at each time in order to maximize the hit rate, i.e., the fraction of requests that result in hits.

III. Analysis of Traditional Caching Policies

In this section, we characterize the performance of traditional caching policies, designed for the setting where the content being cached is not transient. The motivation behind this is to understand the limitations of traditional policies for our setting where requests have stringent freshness requirements and subsequently use these insights to design better caching policies. We first characterize a fundamental limit on the performance of any caching policy in the presence of freshness constraints.

A. Upper Bound on Hit Rate

To provide an upper bound on the hit rate for any policy, we characterize the performance of a more powerful system (System A) where each router has sufficient storage capacity to simultaneously store a measurement from all N producers, i.e., the case when $C = N$.

Lemma 1: Let $h^{(A)}$ be the hit rate in the setting where $C = N$. For the request arrival process in Section II,

$$h^{(A)} \leq \sum_{i=1}^{N} \frac{\lambda_i^2 F_i}{1+\lambda_i F_i}.$$

Proof: Since the cache size is equal to the number of producers in system and there is no benefit in caching more than one measurement of the same producer, we reserve position i in the cache for Producer i's data and study N decoupled cache systems, each of which stores the data of one producer.

To prove the lemma, we map the evolution of each one of the N caches to corresponding M/D/1/1 queues. A request for Producer i at time t is equivalent to an arrival to the corresponding M/D/1/1 queue. A miss at time t is equivalent to a job entering the corresponding M/D/1/1 queue. This job remains in service for the next F_i time-units, i.e., from time t to $t+F_i$, during which all requests for Producer i are blocked because the queue has a buffer of size one. At time $t + F_i$, the job leaves the queue and the next request for Producer i leads to a miss. Since the arrival process is Poisson, by PASTA [11], the hit rate for Producer i is equivalent to the blocking probability for the M/D/1/1 queue.

Using the Erlang-B formula [11] and the insensitivity property [11], the hit rate for Producer i in System A, denoted by $h^{(A,i)} = \frac{\lambda_i F_i}{1+\lambda_i F_i}$. It follows that $h^{(A)} = \sum_{i=1}^{N} \lambda_i h^{(A,i)} = \sum_{i=1}^{N} \frac{\lambda_i^2 F_i}{1+\lambda_i F_i}$. ■

Next, we prove an upper bound on the hit rate in an alternative system (System B) where there are no freshness constraints, i.e., $F_i = \infty \ \forall i$.

Lemma 2: Let $h^{(B)}$ be the hit rate in System B. For the request arrival process discussed in Section II,

$$h^{(B)} \leq \sum_{i \in \mathcal{C}} \lambda_i,$$

where $\mathcal{C}$ is the set of the C most popular producers.

Proof: Since the cache can store the data of at most C producers, the probability of an incoming request leading to a hit is equal to the probability of the request being for one of the C producers whose data is cached. Let S be the set of producers cached at a given time. It follows that

$$h^{(B)} = \sum_{i \in S : |S| \leq C} \lambda_i \leq \sum_{i \in \mathcal{C}} \lambda_i,$$

where $\mathcal{C}$ is the set of the C most popular producers. ■

We use the hit rates in System A and System B to prove the following result.

Proposition 1: Let h_{OPT} be the hit rate of the optimal caching policy for the request arrival process discussed in Section II. Then,

$$h_{\text{OPT}} \leq \min\left\{h^{(A)}, h^{(B)}\right\} = \min\left\{\sum_{i=1}^{N} \frac{\lambda_i^2 F_i}{1+\lambda_i F_i}, \sum_{i \in \mathcal{C}} \lambda_i\right\},$$

where $\mathcal{C}$ is the set of the C most popular producers.

Proof: Refer to [12] for the proof details. ■

B. Store Most Popular (SMP)

Recall that the router has a storage capacity of C units and each producer's data takes one unit of storage. As the name suggests, the SMP policy caches the C most popular contents at the router. To use this policy in our setting where we have stringent freshness requirements, whenever the data stored in the router is not fresh enough to serve an incoming request, a fresh measurement is fetched from the corresponding producer and it replaces the older measurement of the same producer stored at the router. Refer to Algorithm 1 for a formal definition.

Algorithm 1: Store Most Popular (SMP)

```
   Input: The index set $\mathcal{C}$ of $C$ most popular producers,
          freshness requirement of producers in $\mathcal{C}$
1  Initialize: $t_i^{\text{fetch}} = -\infty, \forall i \in \mathcal{C}$
2  On request for Producer $i$ at time $t$ do
3  if $i \in \mathcal{C}$ then
4      if $t_i^{(fetch)} + F_i \geq t$ then
5          Serve request using cached data (cache hit)
6      else
7          Fetch data and serve request (cache miss)
8          Update Producer $i$'s data in cache, $t_i^{(\text{fetch})} = t$
9  else
10     Fetch data and serve request (cache miss)
```

Proposition 2: Let h_{SMP} be hit rate of the SMP policy. For the request arrival process discussed in Section II,

$$h_{\text{SMP}} = \sum_{i \in \mathcal{C}} \frac{\lambda_i^2 F_i}{1+\lambda_i F_i},$$

where $\mathcal{C}$ is the set of the C most popular producers.

Proof: Follows using arguments similar to those used in the proof of Lemma 1. ■

C. Least Recently Used (LRU)

The next policy we study is Least Recently Used (LRU) [13]. The key idea behind the LRU policy is that on a miss, new content is fetched and that cached content which has not been used for the longest time is evicted from the cache to make space to store the fetched content. We use the LRU policy for the setting where requests have stringent freshness requirement by making the following change. If a stale version of the requested producer's data is available in the cache, a fresh measurement is fetched to serve the request and this new data replaces the stale data of the requested producer in the cache. Refer to Algorithm 2 in [12] for a formal definition.

To characterize the performance of the LRU policy for the setting where requests have stringent freshness requirements, we modify the well known Che's approximation [14] which characterizes the performance for LRU in the setting without any freshness constraints. Che's approximation is known to be accurate for the setting without data freshness constraints [15]. The original Che's approximation is as follows.

Approximation 1 (Che's Approximation [14]): Consider an alternative system (System B) where requests have no freshness constraints. Let $h_{\text{LRU}}^{(B,i)}$ be the probability that an incoming request for Producer i in System B leads to a hit. Then, $h_{\text{LRU}}^{(B,i)} \approx 1 - \exp(-\lambda_i T_C^{(i)})$, where $T_C^{(i)}$ is the solution to $C = \sum_{j=1;j\neq i}^{N} 1 - \exp(-\lambda_j T_C^{(i)})$.

Our next result uses Che's approximation to characterize the performance of LRU for caching transient data.

Proposition 3: Let $h_{\text{LRU}}(i)$ be the probability that an incoming request for Producer i leads to a hit under the LRU policy. For the request arrival process discussed in Section II,

$$h_{\text{LRU}}(i) \lesssim \min\left\{h_{\text{LRU}}^{(B,i)}, \frac{\lambda_i F_i}{1+\lambda_i F_i}\right\}$$
$$\implies h_{\text{LRU}} \lesssim \sum_{i=1}^{N} \lambda_i \min\left\{h_{\text{LRU}}^{(B,i)}, \frac{\lambda_i F_i}{1+\lambda_i F_i}\right\}.$$

The approximation can be justified as follows. From the proof of Lemma 1, we have an upper bound on the hit rate for requests for Producer i in System A ($h^{(A,i)}$) where $C = N$, i.e., the cache is large enough to store the data of all producers in the system. For a system with a smaller cache, the hit rate is upper bounded by the hit rate in System A.

From Approximation 1, we have an approximation for the hit rate for Producer i in System B ($h^{(B,i)}$) where there are no freshness constraints. It can be shown by a stochastic coupling argument that the hit rate in the presence of freshness constraints is upper bounded by the hit rate in System B.

D. Random (RAND)

The next policy we study is the Random (RAND) policy. On each miss, the RAND policy fetches the requested content and caches it. As the name suggests, to make space for the fetched content, the RAND evicts one of the current contents of the cache, chosen uniformly at random. Similar to the LRU policy discussed above, we modify the RAND policy to incorporate data freshness constraints. Refer to Algorithm 4 in [12] for a formal definition.

For the setting where requests have no freshness requirements, [16] provides an approximation for the performance

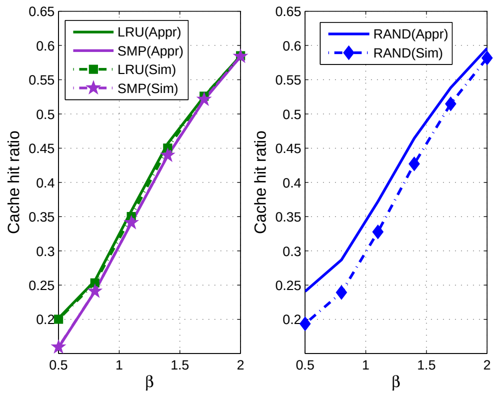

Fig. 3: Hit rate vs Zipf parameter (β) for Zipf popularity for cache size $C = 20$, $N = 100$, $F_A = 5$ and $F_B = 80$.

for the RAND policy. We use this approximation to obtain an approximation for the performance of the RAND policy in the presence of freshness constraints. Refer to [12] for a justification of this approximation.

Proposition 4: Let h_{RAND} be the hit rate for the RAND policy for the request arrival process discussed in Section II. Then,

$$h_{\text{RAND}} \approx \sum_{i=1}^{N} \frac{\lambda_i \mathbb{E}[T_C] - \left\lfloor \frac{\mathbb{E}[T_C]}{F_i + 1/\lambda_i} \right\rfloor}{1 + \lambda_i \mathbb{E}[T_C]},$$

where the value of $\mathbb{E}[T_C]$ is computed by solving the following equation.

$$C = \sum_{j=1}^{N} \frac{\lambda_j \mathbb{E}[T_C]}{1 + \lambda_j \mathbb{E}[T_C]}.$$

E. Accuracy of our Approximations

In this section, we compare the analytical expressions derived in Sections III-B, III-C, and III-D with simulations results to illustrate the accuracy of our theoretical performance guarantees. The simulation results are obtained by computing the empirical hit rates over arrival sequences consisting of 10^7 requests. Refer to [12] for more extensive comparisons between simulated and analytical results. Due to space constraints, in this version, we present results for the following request process. Producer popularity follows Zipf's Law (defined in Section II) with Zipf parameter $\beta > 0$. The freshness requirement for the five most popular producers is denoted by F_A and the freshness requirement for the remaining producers is denoted by F_B.

In Figure 3, we compare the simulation and approximation results for different popularity profiles. In Figure 4, we compare the simulation and approximation results for different freshness requirements. The simulated and analytically computed values are very close to each other, especially for LRU and SMP. The approximations for RAND are comparatively less accurate.

IV. Our Caching Policy: Least Useful

The policies discussed in Section III make caching decisions independent of the residual lifetime of the data, where the residual lifetime of a data is defined as the duration of time

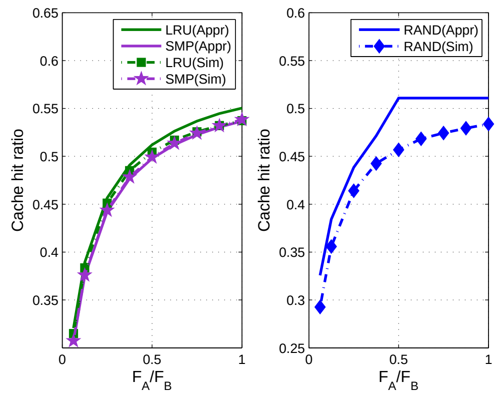

Fig. 4: Hit rate vs freshness requirement for Zipf popularity for cache size $C = 20$, $\beta = 0.8$, $N = 100$.

from the current time to the time when the data will become stale. In this section, we propose a policy called Least Useful (LU) which takes into account the popularity as well as the residual lifetime of each data to make caching decisions.

The key decision a caching policy makes is what to evict from the cache to make space for freshly fetched data. The LU policy evicts the data with the minimum expected number of requests in its residual lifetime. This is motivated by the fact that the data of a popular producer about to go stale could be less useful than the data of a less popular producer with a large residual lifetime because the number of requests served by the latter could be larger. If the expected number of requests in the residual lifetime of the newly fetched data is less than the corresponding values of all data currently in the cache, the fetched content is not stored. Refer to Figure 2 for a formal definition of the LU policy.

Algorithm 2: LEAST USEFUL (LU)

```
Input: Freshness requirement ($F_i$) and popularity ($\lambda_i$)
       of the $N$ producers
1  Initialize $t_i^{(\text{fetch})} = -\infty$, for $1 \leq i \leq N$, $\mathcal{C}_{\text{LU}} = \emptyset$
2  On request for Producer $i$ at time $t$ do
3  if $i \in \mathcal{C}_{LU}$ then
4      if $t_i^{(fetch)} + F_i \geq t$ then
5          Serve request using cached data (cache hit)
6      else
7          Fetch data and serve request (cache miss)
8          Update Producer $i$'s data in cache, $t_i^{(\text{fetch})} = t$
9  else
10     Fetch data and serve request (cache miss)
11     Compute $v_j = \lambda_i(t_j^{(\text{fetch})} + F_j - t), \forall j \in \mathcal{C}_{\text{LU}}$
12     if $\lambda_i F_i > \min_{j \in \mathcal{C}_{LU}} v_j$ then
13         $\mathcal{C}_{\text{LU}} = (\mathcal{C}_{\text{LU}} \setminus \{j^*\}) \cup \{i\}$ where
           $j^* = \arg\min_{j \in \mathcal{C}_{\text{LU}}} v_j$
14         $t_i^{(\text{fetch})} = t$
```

A. Performance Guarantees for the LU Policy

Our next result provides a lower bound on the hit rate of the LU policy.

Proposition 5: Without loss of generality, let the producers be indexed in decreasing order of the product of their popularity and freshness constraints, i.e., $\lambda_1 F_1 \geq \lambda_2 F_2 \geq \cdots \geq \lambda_N F_N$. Let h_{LU} be the hit rate for the LU policy and C be the cache size. For the request arrival process in Section II,

$$h_{LU} \geq \sum_{i=1}^{C-1} \frac{\lambda_i^2 \hat{F}_i}{1 + \lambda_i \hat{F}_i}, \text{ where } \hat{F}_i = F_i - \frac{\lambda_C F_C}{\lambda_i}.$$

Proof: Recall that the LU policy does not store more than one measurement from the same producer. Since producers are indexed in decreasing order of the product of their popularity and freshness constraint, for $i < C$, when Producer i's data is fetched, the product of its popularity and residual lifetime, i.e., $\lambda_i \times F_i$ is more than the popularity-residual lifetime product of at least one of contents currently in the cache. Therefore, for $i < C$, when Producer i's data is fetched, the LU policy stores it in the cache.

Once stored, the data for Producer i for $i < C$ is evicted from the cache only when its popularity-residual lifetime product is the least among the current contents of the cache. Since producers are indexed in decreasing order of the product of their popularity and freshness constraint and the cache can store the data of C producers, the minimum popularity-residual lifetime product among the cached data at any time is upper bounded by $\lambda_C F_C$. Recall that t_i^{fetch} is the time at which the latest measurement was fetched from Producer i. Therefore, if the data of Producer i where $i < C$ is evicted at time t, it follows that

$$\lambda_i(t_i^{\text{fetch}} + F_i - t) \leq \lambda_C F_C \implies t - t_i^{\text{fetch}} \geq F_i - \frac{\lambda_C F_C}{\lambda_i}.$$

Therefore, once fetched and cached, the data of Producer i for $i < C$ stays in the cache for at least $F_i - \dfrac{\lambda_C F_C}{\lambda_i}$ units.

The result then follows using arguments similar to those in the proof of Lemma 1. ■

B. Simulation Results

In this section, we compare the performance of the proposed policy (LU) with policies described in Section III. Refer to [12] for more extensive comparisons. We also compare the performance of the LU policy with the upper bound proved in Section III-A. The simulation results are obtained by computing the empirical hit rates over arrival sequences consisting of 10^7 requests, averaged over 100 iterations. For all the data points reported in this section, the standard deviation is within 2% of the reported average values.

We present results for the following request process. Producer popularity follows Zipf's Law (defined in Section II) with Zipf parameter $\beta > 0$. We divide the producers into three classes, i.e., Class A, Class B and Class D. All producers in Class A, Class B and Class D have freshness requirements of F_A, F_B and F_D respectively with $F_A = F_D$ and $F_B = \gamma \times F_A$, where γ is a constant > 1.

In Figure 5, we plot the performance of caching policies for different cache sizes. In Figure 6, we plot the performance

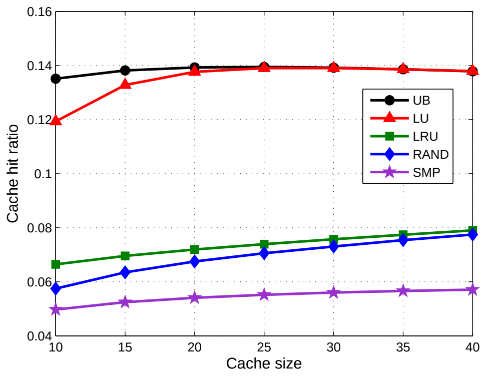

Fig. 5: Hit rate for Zipf popularity for $\beta = 0.8$, $F_A = 5$, $\gamma = 100$, $N = 400$. The LU policy outperforms the LRU, RAND and SMP policies for all values considered. The performance of all four polices improves as the cache size increases.

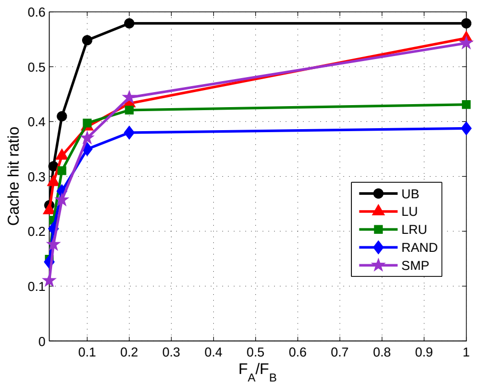

Fig. 7: Hit rate for Zipf popularity for $C = 20$, $\beta = 0.8$, $N = 100$. The LU policy outperforms LRU, RAND and SMP policy.

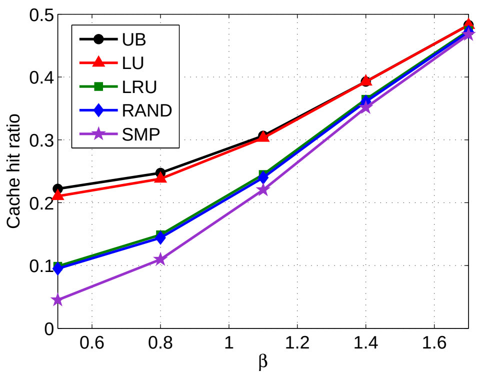

Fig. 6: Hit rate for Zipf popularity for $C = 20$, $F_A = 5$, $\gamma = 100$, $N = 100$. The LU policy outperforms the LRU, RAND and SMP policies for all values considered. The performance of all four polices improves as β increases.

of caching policies for different popularity profiles. In Figure 7, we compare the performance of the caching policies for different freshness requirements. For all cases considered, LU outperforms the traditional policies and its performance is close to the upper bound obtained in Proposition 1. When the freshness requirement is uniform across producers, Figure 7 shows that SMP outperforms LRU and RAND.

V. Conclusions

Motivated by applications like Information Centric Networking for the Internet of Things, we focus on designing caching policies for the setting where the data being cached is transient in nature. We show that traditional caching policies which make eviction decisions agnostic to the residual lifetime of the data being cached are sub-optimal. We characterize the fundamental limitation on the performance of any caching policy for this setting. Next, we propose a new policy which takes into account the residual lifetime of the cached contents in addition to their popularity to make caching/eviction decisions. Via extensive simulations, we show that our policy outperforms traditional caching policies and its performance is close to that of the optimal caching policy.

References

[1] M. Amadeo, C. Campolo, J. Quevedo, D. Corujo, A. Molinaro, A. Iera, R. L. Aguiar, and A. V. Vasilakos, "Information-centric networking for the internet of things: challenges and opportunities," *IEEE Network*, vol. 30, no. 2, pp. 92–100, 2016.

[2] J. Quevedo, D. Corujo, and R. Aguiar, "A case for icn usage in iot environments," in *Global Communications Conference (GLOBECOM), 2014 IEEE*. IEEE, 2014, pp. 2770–2775.

[3] M. A. M. Hail, M. Amadeo, A. Molinaro, and S. Fischer, "On the performance of caching and forwarding in information-centric networking for the iot," in *International Conference on Wired/Wireless Internet Communication*. Springer, 2015, pp. 313–326.

[4] P. Poojary, S. Moharir, and K. Jagannathan, "Caching under content freshness constraints," *arXiv preprint arXiv:1712.10041*, 2017.

[5] S. Vural, P. Navaratnam, N. Wang, C. Wang, L. Dong, and R. Tafazolli, "In-network caching of internet-of-things data," in *Communications (ICC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 3185–3190.

[6] S. Vural, N. Wang, P. Navaratnam, and R. Tafazolli, "Caching transient data in internet content routers," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1048–1061, 2017.

[7] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 2731–2735.

[8] M. Costa, M. Codreanu, and A. Ephremides, "On the age of information in status update systems with packet management," *IEEE Transactions on Information Theory*, vol. 62, no. 4, pp. 1897–1910, April 2016.

[9] R. D. Yates, P. Ciblat, A. Yener, and M. Wigger, "Age-optimal constrained cache updating," in *2017 IEEE International Symposium on Information Theory (ISIT)*, June 2017, pp. 141–145.

[10] C. Kam, S. Kompella, G. D. Nguyen, J. E. Wieselthier, and A. Ephremides, "Information freshness and popularity in mobile caching," in *2017 IEEE International Symposium on Information Theory (ISIT)*, June 2017, pp. 136–140.

[11] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[12] S. Fatale, S. Prakash, and S. Moharir, "Caching polices for transient data," https://www.dropbox.com/s/mw7tziaumbflxg8/main.pdf?dl=0.

[13] A. Dan and D. Towsley, *An approximate analysis of the LRU and FIFO buffer replacement schemes*. ACM, 1990, vol. 18, no. 1.

[14] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1305–1314, 2002.

[15] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for lru cache performance," in *Proceedings of the 24th International Teletraffic Congress*. International Teletraffic Congress, 2012, p. 8.

[16] V. Martina, M. Garetto, and E. Leonardi, "A unified approach to the performance analysis of caching systems," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 2040–2048.