Caching Policies for Transient Data

Santosh Fatale, R 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 life-time (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.**

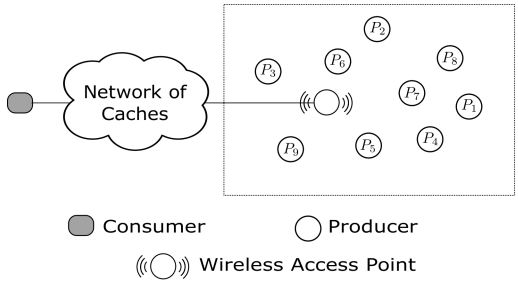

Fig. 1: A system consisting of multiple sensors, a network of routers equipped with caches and consumers.

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) [1]. 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 [2].

Since many IoT applications are information-centric, the idea of using ICN for IoT has gained traction [2]–[4]. The benefits of using ICN for IoT have been explored in [3] where it was found that in-networking caching leads to lower delays and lower power consumption. In [4], 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 intermediate routers as illustrated in Figure 1. Consumers interested in the measurements of these sensors access them via intermediate routers which are 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.

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. In our setting where the data being cached is transient, 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. This motivates the need for new caching policies, specially designed to handle the transient nature of the data being cached. 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 first study a single cache network and show the following.

– We prove a fundamental limit on the performance of any caching policy (Proposition 4) and characterize the performance of traditional caching policies for caching transient data (Propositions 1-2). 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 the performance of the LU policy is close to optimal when the popularity of the producers follows the Zipf distribution and the freshness constraints are uniform across producers (Proposition 6).
– We show that our policy outperforms traditional caching policies via extensive simulations.

In Section III, we extend these results to a two cache network.

B. Related Work

Caching policies for transient data have been studied in [5]–[11]. Closest to this work, in [5], the performance of a similar system 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 arrive according to a Poisson process.

The work in [6] and [7] also focuses on the setting where requests have hard freshness requirements. The key difference between our work and the work in [6] and [7] is the metric used in the two cases. In [6] and [7], for each incoming request, the system pays a fixed convex combination of two costs: *(i)* communication cost – the cost of fetching data from the producer and *(ii)* freshness loss cost – cost proportional to the time elapsed since the data was produced. Due to the second cost, even if the data used to serve the request satisfies the hard freshness constraint, the system still incurs a freshness loss cost. As a result of this, in [6] and [7], even when the cached data can be used to serve a request, the system might prefer to fetch fresh data in order to minimize the overall cost incurred by the system. This model is suitable for highly time-sensitive applications that need real-time data to make decisions.

Contrary to this, in our setting, although we have a hard freshness constraint, for data satisfying this hard constraint, there is zero freshness loss cost. As a result of this, when the cached data is fresh enough to serve a request, the system will never fetch data directly from the producers. In terms of various other modeling differences, we assume that the popularity of various producers is known to the caching policy, whereas [6] and [7] try to estimate this as requests arrive. Given this, a direct comparison between our policy and the policy proposed in [6] and [7] is unfair.

In [8], [9], 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 [8] the focus is on studying the impact of various queuing disciplines like M/M/1, M/D/1, D/M/1. In [9], the authors evaluate the benefits of employing packet management on the system studied in [8]. Like [8] and [9], [10] 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. [11] focuses on caching strategies for the setting where the popularity of a content varies with its freshness. The key takeaway in [11] 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 [11] 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.

C. Organization

The rest of the paper is organized as follows. In Section II, we discuss a single cache system. We first characterize the performance of traditional caching policies and an upper bound on the performance of any policy for our setting where requests have strict freshness requirements. We then propose a new policy and characterize its performance. In Section III we focus on a simple two-cache network. We present our conclusions in Section IV.

II. Single Cache System

A. Setting

We study a system consisting of N sensors (Figure 2), 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.

1) 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 $t - F_i$. In this work, the F_is are fixed and given to us. These values could be 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 that we study in the work 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.,

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

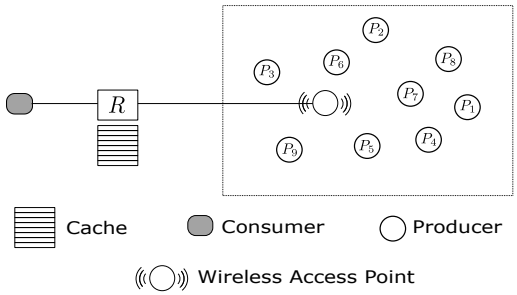

Fig. 2: A system consisting of multiple sensors, a router and consumers. The sensors measure time-varying signals and consumers interested in the measurements of the sensors access them via a router. The router is equipped with storage resources to cache data.

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

2) 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.

3) 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.

4) 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.

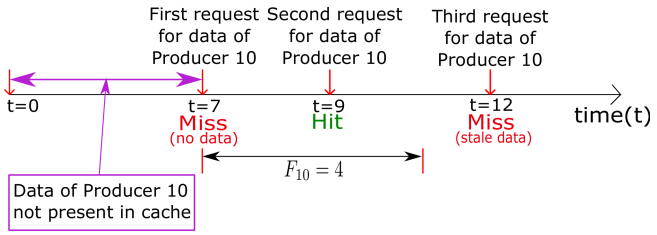

Fig. 3: An illustration of a hit and the two types of misses.

For example (Figure 3), 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/hit ratio, i.e., the fraction of requests that result in hits.

B. 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.

1) 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 $\mathcal{C}$

Initialize: $t_i^{\text{fetch}} = -\infty$, $\forall i \in \mathcal{C}$
On request for Producer i at time t **do**
  if $i \in \mathcal{C}$ **then**
    if $t_i^{(fetch)} + F_i \geq t$ **then**
      Serve request using cached data (cache hit)
    else
      Fetch data and serve request (cache miss)
      Update Producer i's data in cache, $t_i^{(\text{fetch})} = t$
  else
    Fetch data and serve request (cache miss)

Proposition 1: Let h_{SMP} be hit rate of the SMP policy. For the request arrival process discussed in Section II-A,

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

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

Proof: Under the SMP policy, the router stores the C most popular contents and there is no benefit in caching more than one measurement of the same producer, therefore, we reserve position i in the cache for Producer i's data and study C 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 C 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 [12], the hit rate for Producer i is equivalent to the blocking probability for the M/D/1/1 queue.

Using the Erlang-B formula [12] and the insensitivity property [12], the hit rate for Producer i under SMP, denoted by $h_{\text{SMP}}^{(i)} = \frac{\lambda_i F_i}{1+\lambda_i F_i}$. It follows that

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

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

2) Least Recently Used (LRU): The next policy we study is Least Recently Used (LRU) [13]–[16]. 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 for a formal definition.

Algorithm 2: LEAST RECENTLY USED (LRU)

Input: Freshness requirement of the N producers
Initialize: $t_i^{(\text{fetch})} = t_i^{(\text{recent})} = -\infty$, $1 \le i \le N$, $\mathcal{C}_{\text{LRU}} = \emptyset$
On request for Producer i at time t **do**
  if $i \in \mathcal{C}_{LRU}$ **then**
    if $t_i^{(fetch)} + F_i \ge t$ **then**
      Serve request using cached data (cache hit)
      Update $t_i^{(\text{recent})} = t$
    else
      Fetch data and serve request (cache miss)
      Update Producer i's data in cache
      Update $t_i^{(\text{fetch})} = t$, $t_i^{(\text{recent})} = t$
  else
    Fetch data and serve request (cache miss)
    Update $\mathcal{C}_{\text{LRU}} = (\mathcal{C}_{\text{LRU}} \setminus \{j^*\}) \cup \{i\}$, where

$$j^* = \arg\min_{j\in\mathcal{C}_{\text{LRU}}} t_j^{(\text{recent})}$$

    Update $t_i^{(\text{fetch})} = t$, $t_i^{(\text{recent})} = t$

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 [15] 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 [16]. The original Che's approximation is as follows.

Approximation 1 (Che's Approximation [15]): Consider an alternative system (System II) where requests have no freshness constraints and router has a cache of size C units. Let $h_{\text{LRU}}^{(II,i)}$ be the probability that an incoming request for Producer i in System II leads to a hit. Then,

$$h_{\text{LRU}}^{(II,i)} \approx 1 - \exp(-\lambda_i T_C^{(i)}),$$

where $T_C^{(i)}$ is the solution to

$$C = \sum_{j=1;j\neq i}^{N} 1 - \exp(-\lambda_j T_C^{(i)}).$$

To provide an upper bound on the hit rate for any policy, we characterize the performance of a more powerful system (System I) 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 $h^{(I)}$ be the hit rate in the setting where $C = N$. For the request arrival process in Section II-A,

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

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 [12], the hit rate for Producer i is equivalent to the blocking probability for the M/D/1/1 queue.

Using the Erlang-B formula [12] and the insensitivity property [12], the hit rate for Producer i in System I, denoted by

$$h^{(I,i)} = \frac{\lambda_i F_i}{1+\lambda_i F_i}.$$

It follows that

$$h^{(I)} \le \sum_{i=1}^{N} \lambda_i h^{(I,i)} = \sum_{i=1}^{N} \frac{\lambda_i^2 F_i}{1+\lambda_i F_i}.$$

∎

Our next result uses Che's approximation and Lemma 1 to characterize the performance of LRU for caching transient data.

Proposition 2: 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-A,

$$h_{\text{LRU}}^{(i)} \lesssim \min\left\{ h_{\text{LRU}}^{(II,i)}, \frac{\lambda_i F_i}{1+\lambda_i F_i} \right\},$$

and therefore,

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

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 I ($h^{(I,i)}$) 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 I.

From Approximation 1, we have an approximation for the hit rate for Producer i in System II ($h^{(II,i)}$) 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 II.

Since the hit rate in our system is upper bounded by the hit rate is System I and System II, the result follows from the bounds on the hit rate from System I and System II.

3) Random (RAND): The next policy we study is the Random (RAND) policy [16]–[18]. 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. We modify the RAND policy to incorporate data freshness constraints. Refer to Algorithm 3 for a formal definition.

Algorithm 3: RANDOM (RAND)

Input: Freshness requirement of the N producers
Initialize: $t_i^{(\text{fetch})} = -\infty$, $1 \le i \le N$, $\mathcal{C}_{\text{RAND}} = \emptyset$
On request for Producer i at time t **do**
  if $i \in \mathcal{C}_{RAND}$ **then**
    if $t_i^{(fetch)} + F_i \ge t$ **then**
      Serve request using cached data (cache hit)
    else
      Fetch data and serve request (cache miss)
      Update Producer i's data in cache, $t_i^{(\text{fetch})} = t$
  else
    Fetch data and serve request (cache miss)
    Update $\mathcal{C}_{\text{RAND}} = (\mathcal{C}_{\text{RAND}} \setminus \{j^*\}) \cup \{i\}$, where j^* is chosen uniformly at random from the set $\mathcal{C}_{\text{RAND}}$
    Update $t_i^{(\text{fetch})} = t$

For the setting where requests have no freshness requirements, [16] provides the following approximation for the performance for the RAND policy.

Approximation 2: ([16]) Let System II be a system where requests have no freshness requirements and $h_{\text{RAND}}^{(II)}$ be the probability that an incoming request in System II leads to a hit. Let the hit rate for Producer i be denoted by $h_{\text{RAND}}^{(II,i)}$. Then,

$$h_{\text{RAND}}^{(II,i)} \approx \frac{\lambda_i \tau_C}{\sum_{j\neq i} \lambda_j + \lambda_i \tau_C},$$

where τ_C is the solution to

$$C = \sum_{j=1}^{N} h_{\text{RAND}}^{(II,j)} = \sum_{j=1}^{N} \frac{\lambda_j \tau_C}{\sum_{i\neq j} \lambda_i + \lambda_j \tau_C}.$$

The total hit rate is given by

$$h_{\text{RAND}} = \sum_{i=1}^{N} \lambda_i h_{\text{RAND}}^{(II,i)} \approx \sum_{i=1}^{N} \frac{\lambda_i^2 \tau_C}{\sum_{j\neq i} \lambda_j + \lambda_i \tau_C}.$$

We use this approximation to obtain an approximation for the performance of the RAND policy in the presence of freshness constraints.

Proposition 3: Let $h_{\text{RAND}}^{(i)}$ be the hit rate for the Producer i under RAND policy for the request arrival process discussed in Section II-A. Then,

$$h_{\text{RAND}}^{(i)} \lesssim \min\left\{ \frac{\lambda_i \tau_C}{\sum_{j\neq i} \lambda_j + \lambda_i \tau_C}, \frac{\lambda_i F_i}{1+\lambda_i F_i} \right\}$$

and $h_{\text{RAND}} = \sum_{i=1}^{N} \lambda_i h_{\text{RAND}}^{(i)}$, where $C = \sum_{j=1}^{N} \frac{\lambda_j \tau_C}{\sum_{i\neq j} \lambda_i + \lambda_j \tau_C}$.

The approximation can be justified in same manner as of Proposition 2.

4) Accuracy of our Approximations: In this section, we compare the analytical expressions derived in Sections II-B1, II-B2 and II-B3 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. We present results for the following request processes.

- Two-Class Model: Five producers in Class A, $N-5$ producers in Class B. All producers in Class A and Class B have freshness requirements of F_A and F_B time-units respectively. Requests arrive according to a Poisson process. The cumulative probability of an incoming request being for a producer in Class A and Class B is denoted by p_A and $p_B = 1-p_A$ respectively. All producers in each class are equally popular.
- Zipf Popularity: Producer popularity follows Zipf's Law (defined in Section II-A) 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.

The main takeaway from these results is that the simulated and analytically computed values are very close for LRU and SMP for all cases considered. The percentage absolute error for RAND is at most 10%. In Figure 4 and Figure 5, we compare the simulation and approximation results for various cache sizes for both request processes. The approximations for LRU and SMP are close to the the simulation results. Although the approximation for RAND is not as close to the simulation results, the percentage error for both request processes is at most 5%. The increase in the cache hit rate with increase in cache size is very small due to the high values of p_A and β. In Figure 6 and Figure 7, we compare the simulation and approximation results for different popularity profiles for both request processes. The increase in the values of p_A and β assigns higher probability to popular producers, which in turn increases the cache hit rate with p_A and β. For high values of p_A and β, the performance of all policies is very close due to the concentration of the probability mass over a few producers,

which is also visible in Figure 4 and Figure 5. In Figure 8 and Figure 9, we compare the simulation and approximation results for different freshness requirements for both request processes. The horizontal axis represents the ratio of freshness for Class A to the freshness for Class B. The ratio being equal to one implies that all producers have the same (uniform) freshness requirements. As this ratio increases keeping other parameters fixed, the cache hit rate increases. The increase in hit rate occurs due to the increase in the number of hits for the popular producers.

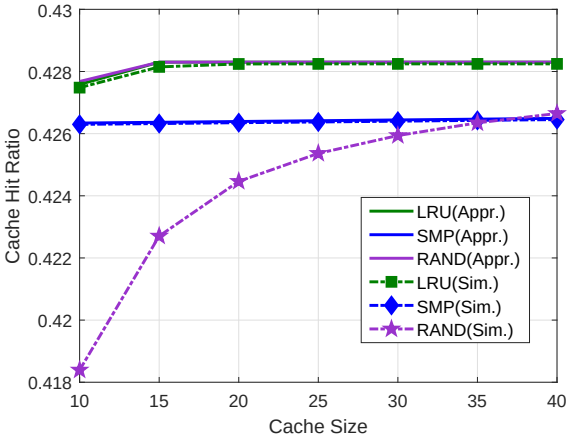

Fig. 4: Hit rate vs cache size for the two-class model with $p_A = 0.9$, $N = 400$, $F_A = 5$, and $F_B = 80$.

C. Upper Bound

In this section, we characterize the fundamental limit on the performance of any caching policy in the presence of freshness constraints.

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

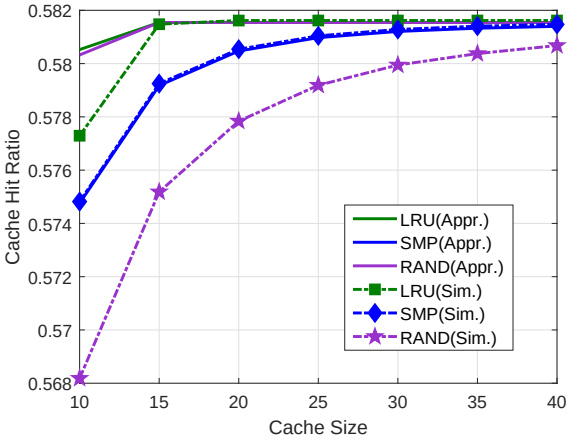

Fig. 5: Hit rate vs cache size for Zipf popularity for $\beta = 2$ and $N = 400$, $F_A = 5$ and $F_B = 80$.

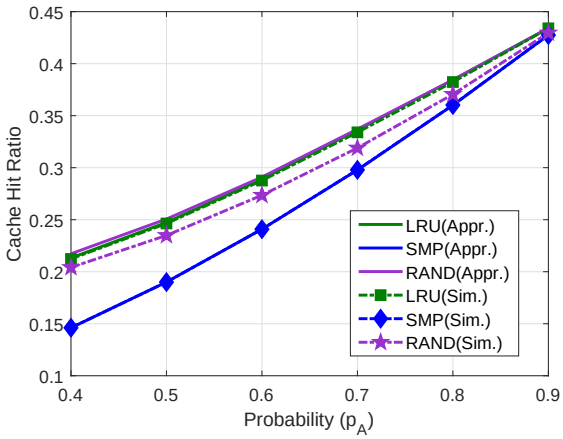

Fig. 6: Hit rate vs popularity for the two-class model for cache size $C = 20$, $N = 100$, $F_A = 5$ and $F_B = 80$.

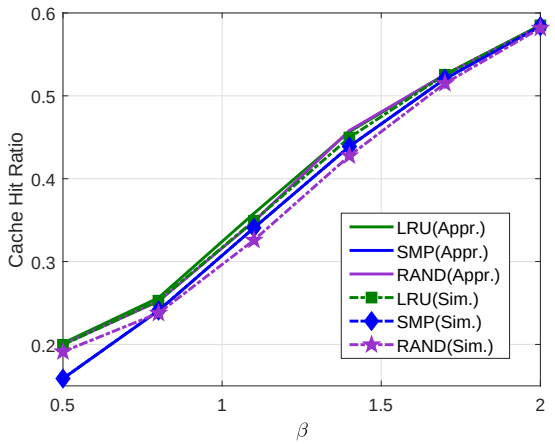

Fig. 7: Hit rate vs Zipf parameter (β) for Zipf popularity for cache size $C = 20$, $N = 100$, $F_A = 5$ and $F_B = 80$.

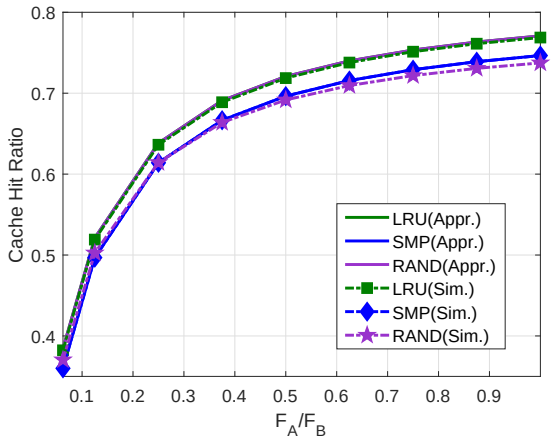

Fig. 8: Hit rate vs freshness requirement for two-class model for cache size $C = 20$, $p_A = 0.8$, $N = 100$.

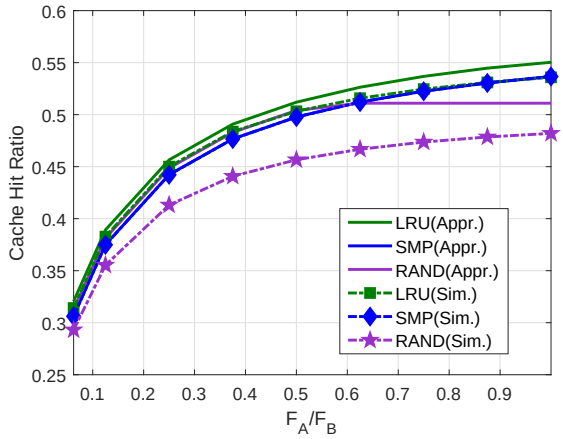

Fig. 9: Hit rate vs freshness requirement for Zipf popularity for cache size $C = 20$, $\beta = 0.8$, $N = 100$.

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

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

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

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

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

Proposition 4: Let h_{OPT} be the hit rate of the optimal caching policy for the request arrival process discussed in Section II-A. Then,

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

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

Proof: From Lemma 1, we have an upper bound on the hit rate in System I ($h^{(I)}$) 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 I.

From Lemma 2, we have an upper bound on the hit rate in System II ($h^{(II)}$) 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 II.

Using these two bounds, it follows that,

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

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

D. Our Caching Policy: Least Useful

The policies discussed in Section II-B 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 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 Algorithm 4 for a formal definition of the LU policy.

Algorithm 4: LEAST USEFUL (LU)

Input: Freshness requirement (F_i) and popularity (λ_i) of the N producers

Initialize $t_i^{(\text{fetch})} = -\infty$, for $1 \leq i \leq N$, $\mathcal{C}_{\text{LU}} = \emptyset$
On request for Producer i at time t **do**
if $i \in \mathcal{C}_{LU}$ **then**
  if $t_i^{(fetch)} + F_i \geq t$ **then**
    Serve request using cached data (cache hit)
  else
    Fetch data and serve request (cache miss)
    Update Producer i's data in cache, $t_i^{(\text{fetch})} = t$
else
  Fetch data and serve request (cache miss)
  Compute $v_j = \lambda_j(t_j^{(\text{fetch})} + F_j - t)$, $\forall j \in \mathcal{C}_{\text{LU}}$
  if $\lambda_i F_i > \min_{j \in \mathcal{C}_{LU}} v_j$ **then**
    $\mathcal{C}_{\text{LU}} = (\mathcal{C}_{\text{LU}} \setminus \{j^*\}) \cup \{i\}$ where $j^* = \arg\min_{j \in \mathcal{C}_{\text{LU}}} v_j$
    $t_i^{(\text{fetch})} = t$

1) 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 discussed in Section II-A,

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

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 $F_i - \frac{\lambda_C F_C}{\lambda_i}$ units.

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

Our next result shows the closeness of the performance of proposed caching policy with the optimal caching policy for the setting where the popularity of various producers follows the Zipf distribution.

Proposition 6: Let N be the number of producers in a system and $C(N) = \omega(1)$[1] be the cache size at the router. Let $F_i = F \ \forall i$ (*i.e.* uniform freshness across producers) and the popularity of the producers follows the Zipf distribution with parameter $\beta > 1$. For a given $\epsilon > 0$, there exists N large enough such that

$$h_{\text{LU}} \geq (1-\epsilon) h_{\text{OPT}},$$

where h_{LU} and h_{OPT} are the hit rates of the LU and the optimal policy respectively.

Proof: For Producer i, from (1),

$$\begin{aligned} h_{\text{LU}}^{(i)} &\geq \frac{\lambda_i F}{1+\lambda_i F}\left(1-(C/i)^{-\beta}\right) \\ &= \frac{\frac{\lambda_1 F}{i^\beta C^\beta}\left(C^\beta - i^\beta\right)}{1+\frac{\lambda_1 F}{i^\beta}}. \end{aligned} \tag{2}$$

Define $\alpha \triangleq \frac{\log\left(1+C(\frac{\epsilon}{2})^{1/\beta}\right)}{\log C}$. It follows that $\exists N$ large enough, such that, for $\alpha < 1$. For Producer i where $1 \leq i < C^\alpha$, from (2),

$$h_{\text{LU}}^{(i)} \geq \frac{\frac{\lambda_1 F}{i^\beta C^\beta}\left(1-\frac{\epsilon}{2}\right)C^\beta}{1+\frac{\lambda_1 F}{i^\beta}} = \left(1-\frac{\epsilon}{2}\right) h_{\text{OPT}}^{(i)}.$$

[1]If $f(n) = \omega(g(n))$ then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

$$\begin{aligned} h_{\text{OPT}} &= \sum_{i=1}^{C^\alpha - 1} \lambda_i h_{\text{OPT}}^{(i)} + \sum_{i=C^\alpha}^{N} \lambda_i h_{\text{OPT}}^{(i)} \\ &= \sum_{i=1}^{C^\alpha - 1} \lambda_i h_{\text{OPT}}^{(i)} + O\left(\frac{1}{(C^\alpha)^{\beta-1}}\right). \end{aligned}$$

Also, $\sum_{i=1}^{C^\alpha-1} \lambda_i h_{\text{OPT}}^{(i)} = \Theta\left(\lambda_1 h_{\text{OPT}}^{(1)}\right)$. It follows that

$$\begin{aligned} h_{\text{OPT}} &= \left(\sum_{i=1}^{C^\alpha-1} \lambda_i h_{\text{OPT}}^{(i)}\right)\left(1+O\left(\frac{1}{(C^\alpha)^{\beta-1}}\right)\right) \\ &\leq \left(\sum_{i=1}^{C^\alpha-1} \frac{\lambda_i h_{\text{LU}}^{(i)}}{\left(1-\frac{\epsilon}{2}\right)}\right)\left(1+O\left(\frac{1}{(C^\alpha)^{\beta-1}}\right)\right) \\ &\leq \frac{1}{\left(1-\frac{\epsilon}{2}\right)} \underbrace{\left(\sum_{i=1}^{N} \lambda_i h_{\text{LU}}^{(i)}\right)}_{h_{\text{LU}}} \left(1+O\left(\frac{1}{(C^\alpha)^{\beta-1}}\right)\right) \\ \implies & \frac{\left(1-\frac{\epsilon}{2}\right)}{1+O\left(\frac{1}{(C^\alpha)^{\beta-1}}\right)} h_{\text{OPT}} \leq h_{\text{LU}}. \end{aligned}$$

For N large enough, $\frac{\left(1-\frac{\epsilon}{2}\right)}{1+O\left(\frac{1}{(C^\alpha)^{\beta-1}}\right)} \geq 1-\epsilon$,

$$\implies h_{\text{LU}} \geq (1-\epsilon) h_{\text{OPT}}.$$

∎

E. Simulation Results

In this section, we compare the performance of the proposed policy (LU) with policies described in Section II-B. We also compare the performance of the LU policy with the upper bound proved in Section II-C. The simulation results are obtained by computing the empirical hit rates over arrival sequences consisting of 10^5 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 processes.

- Three-Class Model: C producers in Class A, C producers in Class B and $N - 2C$ producers in Class D, where C is cache size. All producers in Class A, Class B, and Class D have freshness requirements of F_A, F_B and F_D time-units respectively. Each request is generated according to an i.i.d. process. The cumulative probability of an incoming request being for a producer in Class A, Class B, and Class D is given by p_A, p_B, and p_D respectively. All producers in each class are equally popular.
- Zipf Popularity: Producer popularity follows Zipf's Law (defined in Section II-A) with Zipf parameter $\beta > 0$. We divide the producers in 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.

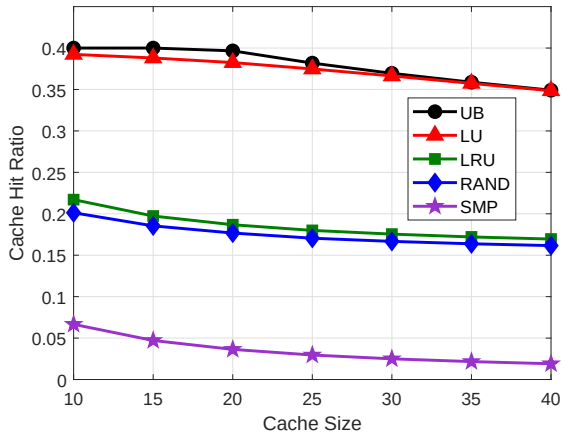

Fig. 10: Hit rate for three class model with $p_A = 0.4$, $F_A = 5$, $\gamma = 100$, $N = 400$. The LU policy outperforms the LRU, RAND and SMP policies for all cases considered. Note that the arrival process changes with the value of the cache size. Therefore, the performance of all the polices deteriorates as the cache size increases.

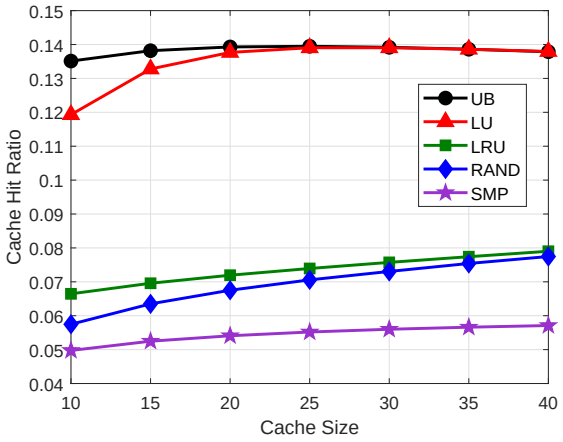

Fig. 11: Hit rate for Zipf popularity for $\beta = 0.8$, $F_A = 5$, $\gamma = 100$, $N = 400$. The LU policy outperforms the LRU, RAND and SMP policies for all values considered. The performance of all four polices improves as the cache size increases.

The key takeaway from the results is that for all cases considered, LU outperforms the traditional policies and its performance is close to the upper bound obtained in Proposition 4. In Figure 10 and Figure 11, we plot the performance of various caching policies with the upper bound for the two request processes as a function of cache size. In Figure 10, the deterioration in performance of all policies with cache size is due to the change in the arrival process with cache size for Three-Class model. In Figure 11 we observe that the cache hit rate increase with the cache size. In Figure 12 and Figure 13, we plot the performance of caching policies with the upper bound for different popularity profiles for the two request processes. The cache hit rate increases with the cumulative probability of Class A producers and the Zipf parameter, β. For high values of β, all policies perform well (Figure 7). In Figure 14 and Figure 15, we compare the performance of the caching policies with the upper bound for different freshness requirements for the two request process. When the freshness requirement is uniform across producers, Figure 14 and Figure 15 show that SMP outperforms LRU and RAND.

III. Extension to a Simple Cache Network

In this section we focus on a simple cache network shown in Figure 16. The consumer and producers are connected via Routers 1 and 2. All request arrive at R_1. The request arrival process is identical to the single cache setting discussed in Section II. If there is a miss at R_1, the request is forwarded to R_2. Router R_2 has direct access to the producers. If miss occurs at R_2 as well, fresh data is fetched from the producers to serve the request. Let the cache sizes at R_1 and R_2 be C_1 and C_2 respectively.

In this section, we refer to the event of an incoming request being served by a measurement stored at any one of the routers in the network as a *hit* and the complementary event as a *miss*. The goal is to design caching policies to maximize the hit rate.

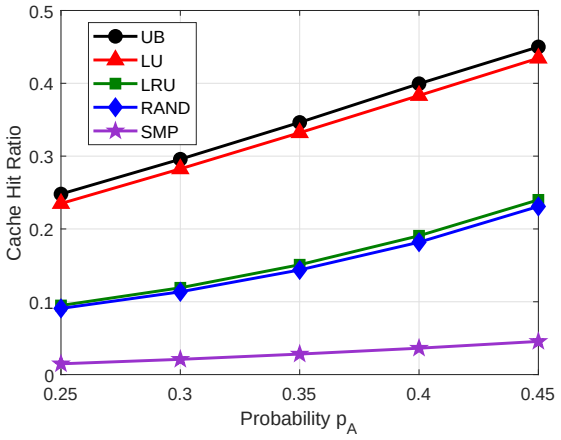

Fig. 12: Hit rate for three class model for $C = 20$, $F_A = 5$, $\gamma = 100$, $N = 100$. The LU policy outperforms the LRU, RAND and SMP policies for all values considered.

A. Traditional Caching Policies

In this section, we characterize the performance of traditional caching policies for the Tandem network.

1) Store Most Popular (SMP): The hit rate at a node is calculated by extending the results for the single cache setting discussed in Section II-B1.

Proposition 7: Let h_{SMP} be hit rate of the SMP policy, then

$$h_{\text{SMP}} = \sum_{i \in \mathcal{C}_1 \cup \mathcal{C}_2} \frac{\lambda_i^2 F_i}{1 + \lambda_i F_i}.$$

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

2) Least Recently Used: In [18], an approximation for the hit rate under the LRU policy for multiple caches was presented for the setting where requests had no freshness constraints. Next, we extend this approximation to our setting.

Proposition 8: Let $h_{\text{LRU}}^{(i)}(j)$ be hit rate at router R_j for Producer i for the Tandem network.

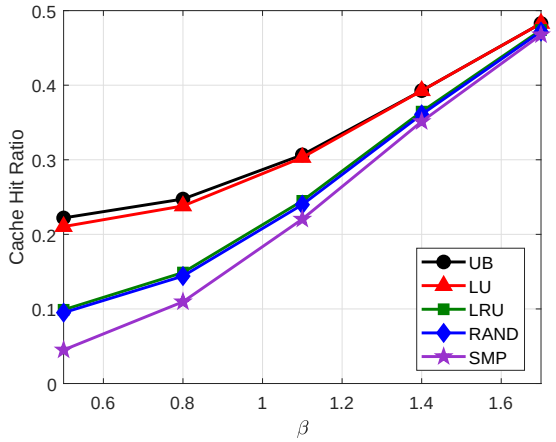

Fig. 13: Hit rate for Zipf popularity for $C = 20$, $F_A = 5$, $\gamma = 100$, $N = 100$. The LU policy outperforms the LRU, RAND and SMP policies for all values considered. The performance of all four polices improves as β increases.

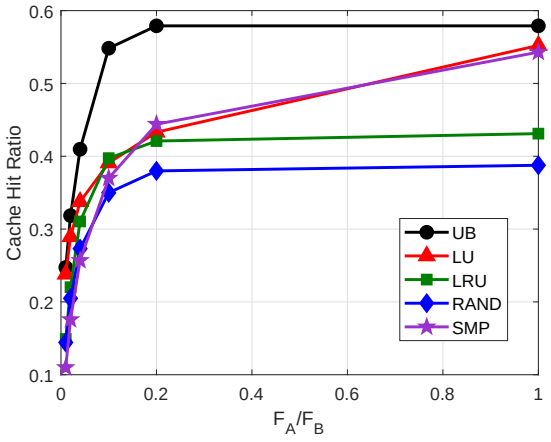

Fig. 15: Hit rate for Zipf popularity for $C = 20$, $\beta = 0.8$, $N = 100$. The LU policy outperforms LRU, RAND and SMP policy. The performance of the SMP policy improves and approaches the performance of the LU policy as the freshness requirements for all sensors approach the same value.

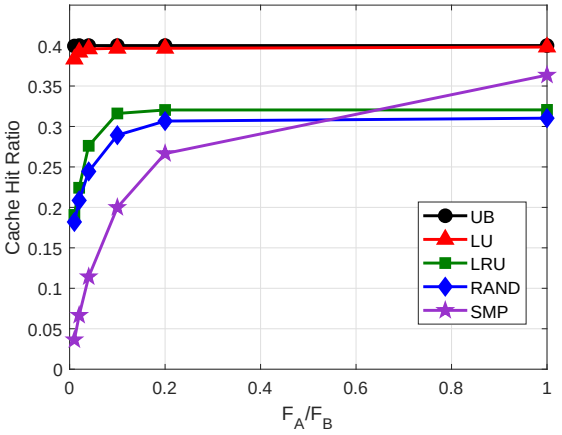

Fig. 14: Hit rate for three class model for $C = 20$, $p_A = 0.4$, $N = 100$. The LU policy outperforms the LRU, RAND and SMP policies for all value considered. The performance of the SMP policy improves as the freshness requirements for all sensors approach the same value.

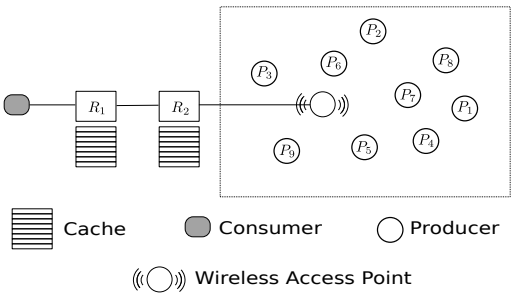

Fig. 16: Tandem network

(i) $h^{(i)}_{\text{LRU}}(1) \approx \min\left\{\tilde{h}^{(i)}_{\text{LRU}}(1), \frac{\lambda_i F_i}{1+\lambda_i F_i}\right\}$, where $\tilde{h}^{(i)}_{\text{LRU}}(1) = 1 - e^{-\lambda_i T^{(i)}_{C_1}}$ and $T^{(i)}_{C_1}$ is the solution to $C_1 = \sum_{j=1;j\neq i}^{N} \tilde{h}^{(j)}_{\text{LRU}}(1)$.

(ii) $h^{(i)}_{\text{LRU}}(2) \approx \min\left\{\tilde{h}^{(i)}_{\text{LRU}}(2), \frac{\lambda^{\text{LRU}}_i(2) F_i}{1+\lambda^{\text{LRU}}_i(2) F_i}\right\}$, where $\lambda^{\text{LRU}}_i(2) = \lambda_i(1 - h^{(i)}_{\text{LRU}}(1))$, $\tilde{h}^{(i)}_{\text{LRU}}(2) = 1 - e^{-\lambda^{\text{LRU}}_i(2) \max\left\{0, \left(\min\left\{F_i, T^{(i)}_{C_2}\right\} - T^{(i)}_{C_1}\right)\right\}}$, and $T^{(i)}_{C_2}$ is the solution to $C_2 = \sum_{j=1;j\neq i}^{N} \tilde{h}^{(j)}_{\text{LRU}}(2)$.

Let h_{LRU} be the hit rate of LRU policy for the Tandem network.

$$h_{\text{LRU}} \approx \sum_{i=1}^{N} \lambda_i h^{(i)}_{\text{LRU}}(1) + \lambda^{\text{LRU}}_i(2) h^{(i)}_{\text{LRU}}(2).$$

This approximation uses ideas from [18] and is justified as follows. Consider the event of a hit at Cache 2. Since a hit occurs at Cache 2 only when there is a miss at Cache 1, the effective request arrival rate at Cache 2 for Producer i is

$$\lambda^{\text{LRU}}_i(2) = \lambda_i(1 - h^{(i)}_{\text{LRU}}(1)).$$

Similar to [18], we approximate the arrival process at Cache 2 with a Poisson process with this rate.

Provided $\min\{F_i, T^{(i)}_{C_2}\} > T^{(i)}_{C_1}$, there will be a hit at Cache 2 at time t for a request for Producer i only if there is at least one request that arrives at Cache 2 in the interval $[t - \min\{F_i, T^{(i)}_{C_2}\}, t - T^{(i)}_{C_1}]$. If $\min\{F_i, T^{(i)}_{C_2}\} \leq T^{(i)}_{C_1}$, $\tilde{h}^{(i)}_{\text{LRU}}(2) = 0$. This gives the result.

Next, we illustrate the accuracy of our approximations for the performance of the SMP and LRU policies for the Tandem network. We focus on the setting where producer popularity follows the Zipf distribution with parameter β. We consider four settings with the following freshness constraints.

Definition 1: (Freshness constraints) Let the freshness constraints for producer be denoted by F_i.

Case 1 ($F = 500$): $F_i = 500$, $\forall i$.
Case 2 ($F = 30$): $F_i = 30$, $\forall i$.
Case 3 ($F = 5$): $F_i = 5$, $\forall i$.

Case 4 (F_{3b}):

$$F_i = \begin{cases} 500 & \text{if } 11 \leq i \leq 20, \\ 5 & \text{otherwise.} \end{cases}$$

In Figure 17, we compare the simulated performance of the SMP policy with the approximation in Proposition 7 for a system consisting of 100 producers, i.e., $N = 100$.

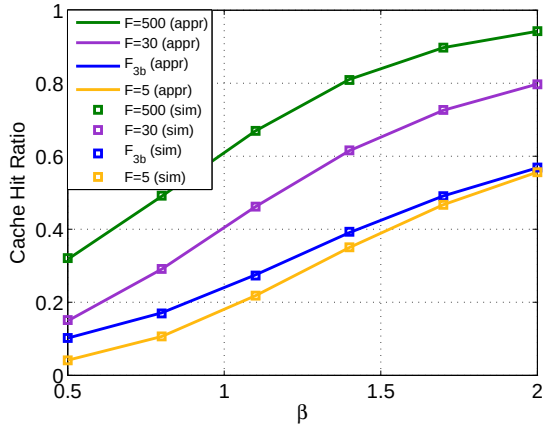

Fig. 17: Comparison between the simulated performance of the SMP policy with the approximation in Corollary 7 for the Tandem network for four different freshness constraint settings. We plot the hit ratio as a function of the Zipf parameter β for $C_1 = 10$ and $C_2 = 15$.

In Figure 18, we compare the simulated performance of the LRU policy with the approximation in Proposition 8 for a system consisting of 100 producers, i.e., $N = 100$.

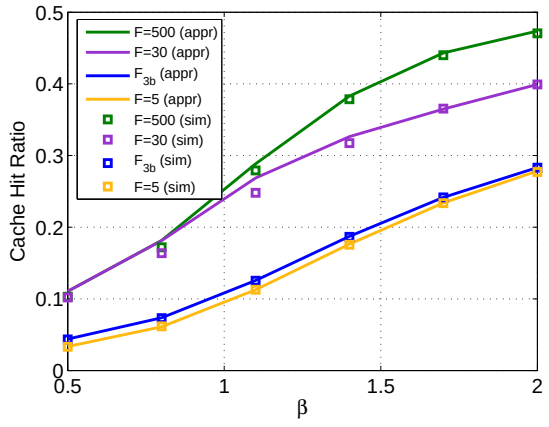

Fig. 18: Comparison between the simulated performance of the LRU policy with the approximation in Proposition 8 for the Tandem network for four different freshness constraint settings. We plot the hit ratio as a function of the Zipf parameter β for $C_1 = 10$ and $C_2 = 15$.

B. Upper Bound on Hit Rate

Proposition 9: Let h_{OPT} be the hit rate of the optimal caching policy, Producer i is requested with probability p_i. Then, we have that,

$$h_{\text{OPT}} \leq \min\left\{ \sum_{i=1}^{N} \frac{p_i \lambda_i F_i}{1 + \lambda_i F_i}, \sum_{i \in \mathcal{C}} p_i \right\},$$

where $\mathcal{C}$ is the set of the $C_1 + C_2$ most popular producers.

Proof: By Lemma 2,

$$h_{\text{OPT}} \leq \sum_{i \in \mathcal{C}} p_i.$$

We upper bound the hit rate of this system by the hit rate of a system with infinite cache memory. By Lemma 1,

$$h_{\text{OPT}} \leq \sum_{i=1}^{N} \frac{p_i \lambda_i F_i}{1 + \lambda_i F_i},$$

We use the two upper bounds on hit rate to conclude that

$$h_{\text{OPT}} \leq \min\left\{ \sum_{i=1}^{N} \frac{p_i \lambda_i F_i}{1 + \lambda_i F_i}, \sum_{i \in \mathcal{C}} p_i \right\}.$$

■

C. Cooperative Caching Policy for a Tandem Network

In this section we propose a caching policy for the Tandem Network which take the freshness of the data being cached into account. In addition, we allow for cooperation across caches in deciding which contents to cache, *i.e.*, there is signaling between the routers to exchange the information. This information is used to make joint caching decision. We ignore the time delays in data transfer from one cache to another.

Under our cooperative caching policy, we implement the Least Useful policy while ensuring there is no common data in the caches at the two routers. Since R_1 receives new data only through R_2, R_2 makes decisions about what R_1 should cache in the following manner. When new data is fetched, R_2 compares the popularity-residual lifetime product of the new data with the popularity-residual lifetime product of all the data currently cached at R_1 and R_2. The newly fetched data replaces the data with the minimum popularity residual lifetime product across the two caches if there exists at least one data with a lower popularity-residual lifetime product than the newly fetched data.

In Figure 19, we compare the performance of our cooperative LU policy with other non-cooperative caching policies (SMP, LRU and RAND implemented independently at all caches) and the upper bound on the hit rate. We consider a network consisting of 100 producers, i.e., $N = 100$ with requests arriving at Router 1 according to a Poisson process with rate one. The popularity of the producers follows the Zipf distribution with parameter β. The cache sizes at Routers 1 and 2 are 10 and 15 respectively. The freshness constraints on the N producers are as in Case 4 in Definition 1. We see that our policy outperforms the three traditional caching policies which do not take the freshness of data into account. The performance of all policies increase with β due to concentration of probability mass over first few producers with increase in β. For high value of β, all policies are able to store data of same set of producers.

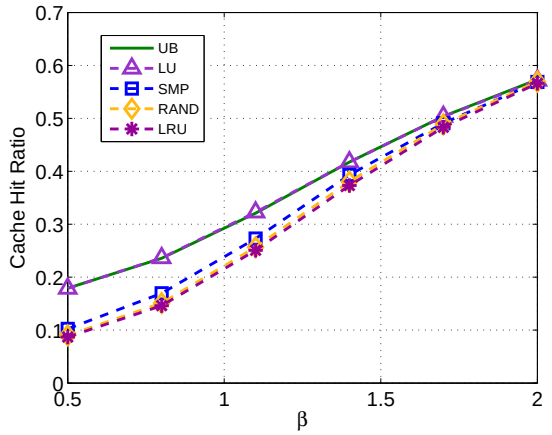

Fig. 19: Comparison between our cooperative LU policy with other non-cooperative caching policies (SMP, LRU and RAND implemented independently at all caches) and the upper bound on the hit rate for the Tandem network for freshness constraint as in Case 4. We plot the hit ratio as a function of the Zipf parameter β for $C_1 = 10$ and $C_2 = 15$.

IV. 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. For the single cache setting and a two cache setting, 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 limits on the performance of any caching policy for both settings. Next, for both settings, 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 the proposed policy outperforms traditional caching policies and its performance is close to that of the optimal caching policy.

References

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, July 2012.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

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

[14] W. King, "Analysis of paging algorithms," in *Proc. IFIP 1971 Congress, Ljubljana*. North-Holland, 1972, pp. 485–490.

[15] 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.

[16] 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.

[17] E. Gelenbe, "A unified approach to the evaluation of a class of replacement algorithms," *IEEE Transactions on Computers*, vol. C-22, no. 6, pp. 611–618, June 1973.

[18] 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.