IEEE ComSoc IEEE Communications Society | IEEE COMPUTER SOCIETY | IEEE Signal Processing Society | VTS Connecting the Mobile World | IEEE Transactions on Machine Learning in Communications and Networking

Received 5 March 2023; revised 17 October 2023 and 28 June 2024; accepted 28 July 2024.
Date of publication 31 July 2024; date of current version 7 August 2024.

The associate editor coordinating the review of this article and approving it for publication was A. El Gamal.

Digital Object Identifier 10.1109/TMLCN.2024.3436472

On Learning Suitable Caching Policies for In-Network Caching

STÉFANI PIRES[1,2], ADRIANA RIBEIRO[1], AND LEOBINO N. SAMPAIO[1]

[1]Department of Computer Science, Federal University of Bahia, Salvador 40170-110, Brazil
[2]Federal Institute of Bahia, Salvador 40301-015, Brazil

CORRESPONDING AUTHOR: S. PIRES (stefani.pires@ifba.edu.br)

This work was supported in part by the National Council for Scientific and Technological Development (CNPq) under Grant 402854/2022-5 and Grant 316208/2021-3; in part by the Bahia State Research Support Foundation (FAPESB) under Grant TIC0004/2015, Grant BOL0023/2020, and Grant BOL0428/2021; and in part by the Air Force Office of Scientific Research under Award FA9550-23-1-0631.

ABSTRACT In-network cache architectures, such as Information-centric networks (ICNs), have proven to be an efficient alternative to deal with the growing content consumption on networks. In caching networks, any device can potentially act as a caching node. In practice, real cache networks may employ different caching replacement policies by a node. The reason is that the policies may vary in efficiency according to unbounded context factors, such as cache size, content request pattern, content distribution popularity, and the relative cache location. The lack of suitable policies for all nodes and scenarios undermines the efficient use of available cache resources. Therefore, a new model for choosing caching policies appropriately to cache contexts on-demand and over time becomes necessary. In this direction, we propose a new caching meta-policy strategy capable of learning the most appropriate policy for cache online and dynamically adapting to context variations that leads to changes in which policy is best. The meta-policy decouples the eviction strategy from managing the context information used by the policy, and models the choice of suitable policies as online learning with a bandit feedback problem. The meta-policy supports deploying a diverse set of self-contained caching policies in different scenarios, including adaptive policies. Experimental results with single and multiple caches have shown the meta-policy effectiveness and adaptability to different content request models in synthetic and trace-driven simulations. Moreover, we compared the meta-policy adaptive behavior with the Adaptive Replacement Policy (ARC) behavior.

INDEX TERMS In-network caching, cache replacement policy, online learning, multi-armed bandits, adaptive system.

I. INTRODUCTION

In recent years, the proliferation of bandwidth-needed applications and the increased capacity of modern communication devices (e.g., smartphones, network-equipped vehicles, wearables) have led to a bloom of multimedia content consumed at the network. Due to this emerging scenario, the host-centric Internet model has experienced significant challenges in meeting the current and future users' and applications' requirements. One of the strategies to make the Internet feasible in such high content distribution scenarios relies on networking caching approaches, which use cache-equipped devices closer to users to provide the most frequently requested content locally. Information-centric Networking (ICN) [1] is a promising initiative that implements in-networking caching, replicating content in a distributed way in cache-enable routers (CRs) over the network. It benefits users through better quality of service levels and operators by saving network resources.

Caching on the network leverages the content delivery process and improves network performance. However, the caching benefits will be efficiently achieved only by deploying caching policies suitable to the network context in which the cache operates. The attention to the caching policies is of paramount importance, especially in modern networks with the recent advances in 5th generation (5G) technology, Mobile Edge Computing (MEC), and network virtualization. Such technologies are revolutionizing the edge, allowing the emergence of new content-demanding applications for

the Internet of Things (IoT), Vehicular Ad Hoc Networks (VANETs), and new network types. The ICN model natively supports those new environments' mobility, scalability, and security requirements. In this way, there is an increasing tendency to deploy ICN-enable edge networks in 5G-ICN virtual network slices [2], [3]. That scenario allows the dynamic creation/relocation of virtual cache nodes according to the demand for content consumption and customized for the cache operation context.

One relevant challenge in creating customized cache instances is choosing which cache replacement policy should be instantiated to achieve optimal caching performances. Traditionally, the choice of caching policies requires some empirical knowledge associating context characteristics of the cache production environment with the policy type that should be adopted. For example, some works point to recency-based policies as more suitable for traffic with strong temporal correlation, such as video streaming [4], or frequency-based policies for more stationary traffic patterns [5]. Other works point to random policies as an efficient choice to handle filtered traffic from a hierarchical caching structure [6]. However, it is not always possible to know the workload scenario's characteristics. Besides, traffic characteristics can change over time, leading to changes in which caching policy would better fit the cache. Changes in the number of users, predominant user habits, and application type (among other contexts) can dynamically change the characteristics of the network at different times. Also, user mobility imposes a challenge leading to on-demand changes in the network topology, node connectivity, and workload aspects. Variations in context aspects may affect policies' performance over time, and the cache must adapt to those changes to ensure optimal performance.

The ICN literature presents a variety of caching policies types [7], [8] exploring different aspects, such as content frequency, recency, size, age, network delay, or energy cost to store the content. Most are static policies and cannot deal with dynamic changes affecting the cache efficiency. Some initiates, such as the Adaptive Replacement Policy (ARC) [9], attempt to tackle that limitation by proposing adaptive policies capable of adjusting the policy's behavior. However, there is no explicit pattern defining which policy is most suitable for a given environment since several factors can influence network traffic and, consequently, the policies' performance. As the current literature has shown, there is no single caching policy to suit the needs of all operation contexts since the policies' performance may vary according to unbounded aspects. Even adaptive policies, like ARC, may not suit appropriately in all scenarios [7]. ICNs can have improved performance by creating customized caches concerning the context. Still, we currently lack mechanisms to help choose suitable caching policies according to the cache operation context in a customized way. That is our focus.

We introduce a caching meta-policy strategy that enhances the cache with the ability to learn the suitable policy among a fixed set of candidate policies without prior knowledge about the cache production environment [10]. The strategy benefits from the diversity of existing caching policies and models the choosing process as an online learning with a partial feedback problem. The meta-policy decouples the replacement decision reasoning from the context information used by the policies to enable learning with multiple policies. The meta-policy is a generic and modular strategy, allowing the cache to learn according to the operational environment. The central point is to enable learning at an upper abstraction level instead of directly making content replacement decisions. It's not how the cache learns since the learning profile is to be carried by a particular learning agent instance. Also, it is not the delimitation of the candidate policies, as it depends on the degree of diversity required in a particular implementation.

As a proof-of-concept evaluation, we have implemented the strategy in Named-data networking (NDN) [11] caching nodes, with a sample set of standard caching policies and variations of non-stationary stochastic bandits playing the learning agent. Then, we conducted simulation-based experiments through multiple scenarios with synthetic and real web traces. The results show that the cache node enhanced with the meta-policy strategy approximates the best heuristic replacement policy's performance according to each scenario without prior knowledge about the production environments. The strategy allowed the cache to work as an adaptive system, learning the suitable policies for different network situations. To further assess the adaptive behavior, we have also implemented the ARC policy to compare the ARC adaptive behavior to the behavior of our strategy. The adaptive behavior achieved by the meta-policy relies on the diversity of the caching policy set. The advantage is that adaptability can be comprehensively customized. One can explore a broad and unbounded scope of context information by adding policies in the set, including adaptive policies. Therefore, the adaptability achieved by the meta-policy has a broader range of applications by encompassing adaptive policies.

The main contributions of this paper are:

- Modeling the choice of optimal caching policies as online learning with partial feedback problems, in which cache-enabled routers can employ different cache replacements policies over time. The model can be applied in single and multi-cache networks.
- Providing a modular meta-policy approach to assist the choosing process of suitable caching policies according to the current context, and further, to cope with the natural dynamism of context variations in networks. The meta-policy strategy is generic regarding the network type in which the cache operates.
- Experimentally evaluating the meta-policy in NDN scenarios relying on simulations using both synthetic and real traces.
- Comparing the adaptive behavior of the meta-policy with the ARC policy. Our method works as an adaptive system by adjusting its choice of the best policy for a cache in operation. The choice is among a finite set of

existing caching policies. Although the adaptation process differs, one adaptive policy has the closest expected results. Therefore, we have performed experiments to compare our method with ARC. The experiments include using ARC as one of the candidate policies employed by the learning agent.
- Discussing the applicability of our meta-policy approach to other caching systems and the impact on implementing caching policies.

The remainder of the paper is organized as follows: Section II presents background and discusses related work. Section III presents our system model, and section IV details the caching meta-policy strategy. Section V contains the evaluation methodology, and section VI presents the results along with remaining remark discussions. Section VII concludes the paper and discusses future work.

II. BACKGROUND AND RELATED WORK

A. CACHING IN ICNs

In-network caching is a core feature of ICN models. It refers to the mechanisms by which content is cached and managed within the network infrastructure itself, typically in the CRs. The content's name is an essential element for network routing in ICNs. The content requests are forwarded based on the name of the content, not its location, which enables the decoupling of content location from the content delivery process. To this end, ICN replicates content in a distributed way in the CRs. One approach to disseminating content in ICNs is called the *on-path* approach. As content flows through the network from source to destination, routers along the content delivery path can cache copies of the content they encounter. The on-path in-network caching is an opportunistic manner to distribute content over the network.

The CRs can operate with different policies to tackle cache management. The policies can be classified as content placement and replacement policies. Placement policies focus on deciding whether passing content should be stored locally. They can operate isolated or in a cooperative way. Examples of placement policies include Leave Copy Everywhere (LCE), Probabilistic caching (Prob), and Leave Copy Down (LCD) [12]. Meanwhile, content replacement policies define the logic used to choose which content to evict from the cache when there is the need for storing new content and no more space is available in the cache. This work focuses on the replacement policies.

Caches in ICNs can work with traditional replacement policies, such as Least Recently Used (LRU), Least Frequently Used (LFU), First-In-First-Out (FIFO) and Random, or newly proposed context-specific policies that explore a diversity of context information in their eviction logic [8]. Each policy uses a different logic to elect which content is less likely to be accessed in the near future. For example, LRU removes the content with the oldest access date; LFU elects to remove the content with the least amount of access; FIFO selects the content stored for longer in the cache; and Random removes any content randomly. Table 1 summarizes the context information used by these basic policies.

TABLE 1. Basic cache replacement policies and the context information used in their eviction logic.

Policy	Context Information
LRU	Content last access time
LFU	Content access frequency
FIFO	Content arrival time in the cache.
Random	–

Other replacement policies explore a variety of context aspects involving not just content-related information but also cache node and network-related data. For instance, the replacement policy proposed in [13] is content and node-based, using in its eviction logic the content producer identification, the content creation time on the producer, and the total number of chunks/content by producers in the cache. Reference [8] contains a comprehensive list of different context information used by replacement policies in ICNs.

B. ONLINE LEARNING FOR CACHING SYSTEMS

Online learning models represent a subset of machine learning techniques to tackle online prediction problems. An agent interacts with the environment in successive online rounds by taking one action at each round over a range of possibilities. Different actions cause different impacts on the environment measured by corresponding reward values. Each iteration is a new decision instance, and the problem is to predict which action to execute, aiming to maximize the cumulative rewards in a time horizon (or reduce the regret when not choosing the best action). The model of choosing over a finite set of actions is known as learning from expert advice, as each action plays as an expert by returning numerical feedback information used to improve future predictions. Two usual feedback models are the *full information*, in which the agent has access to the feedback of all actions, and the *partial information*, in which only the played action yields its feedback value. This partial feedback setting is also known as bandit feedback.

Bandit models are widely explored to solve resource allocation problems in caching systems [14], [15], [16], [17], [18], [19]. In particular, the content allocation problem in cache-enabled cellular infrastructures can be modeled as bandit applications. For instance, [14] proposed a bandit model for content placement in wireless small base stations (sBS). The model places an agent as the manager of an sBS and the contents present in a connected macro base station (mBS) as the actions. The bandit problem is to select a subset of popular contents to cache at the sBS by accounting for the popularity of the contents present in the sBS only. The agent has no information about the popularity of all contents placed at the mBS. This way, the strategy must explore to choose the best subsets of popular content.

Moreover, the authors in [20] employ an online gradient ascent method as caching policy to address the content allocation problem. Gradient ascent is a type of optimization algorithm also used to solve resource allocation problems. Following the content allocation problem's general goal, the strategy chooses a smaller subset of potential popular contents to store at the cache given a content set. Each arriving content request triggers the proposed algorithm to adapt the cached subset based only on the current cache composition and the most recent request.

Another study in [21] models cache content configurations as cache states and transition states as Markov. The work shapes caching policies as online distribution learning algorithms, in which each caching policy can be associated with a distinct popularity distribution of the cached contents. The authors also propose an adaptive policy directed to the learning process under non-stationary request models based on the study.

The employment of online learning techniques as a content replacement policy can be computationally costly since the cache has to trigger the learning agent at each content request arriving at the cache. Besides, the commonality of those and most proposed cache-related solutions is the focus on the content choice. Naturally, that is the central objective of the caching policies. We, instead, model the online caching problem with an upper abstraction level. We propose employing online learning to enhance the cache with the meta capacity to choose among an available set of potentially suitable policies. Our model overcomes the intensive iteration of learning agents as the agent needs to interact with the cache in predefined time intervals, instead of at each arriving content request. This setting is particularly conducive for in-network caching architectures with no restrictions on the content set.

In machine learning language, a sequence of actions chosen according to some learning algorithm is called *policy*. This work applies the term policy only to refer to the caching replacement strategies to avoid misunderstanding.

III. SYSTEM MODEL

Consider a cache-enabled router CR with fixed capacity for n contents from a content library set L of unknown size, but it is known that $|L| \gg n$. The router responds to the content requests passing the network when the content is stored locally, thus counting a cache hit. Otherwise, the router forwards the request to another node on the network and counts a cache miss. Content packets passing through the cache can be opportunistically stored locally, but the cache space is always fully occupied in the steady-state. Therefore, the cache works with a cache replacement policy ω to keep the contents most likely to be reaccessed. That model represents the on-path in-network caching approach of ICNs.

Given a discrete-time setting, we slotted the time into fixed intervals I. The cache efficiency of CR inside the interval I can be defined as:

$$CE(\omega)_I = \frac{H_I}{M_I + H_I} \tag{1}$$

in which H_I is the number of content interests satisfied by the cache in I and M_I is the number of missed requests in the same interval. The cache efficiency relies on the policy ω since different policies perform differently according to their eviction logic.

Consider a finite set $\Omega = (\omega_1, \omega_2, \omega_3, \ldots, \omega_m)$ of m content replacement policies feasible to the CR, for $m \in \mathbb{N}$. Without loss of generality, we assume that $CE(\omega_1)_I > CE(\omega_2)_I > CE(\omega_3)_I > \ldots > CE(\omega_m)_I$. Each $\omega \in \Omega$ can use a different set of contextual information for the eviction logic. Examples of context information includes content access frequency, content last access time, content type, priority or monetary cost, node's connectivity, hop count from the cache node to the content producer, or network delay for retrieving the content [8].

We denote the best policy choice for CR in time interval I as

$$\omega_I^* = \omega_i \mid CE(\omega_i)_I \geq CE(\omega_j)_I, \forall j \leq m. \tag{2}$$

Network environments are dynamic, and variations in the content request pattern can lead to variations in policies' performances. Therefore, the best policy may unpredictably vary over time. The problem is choosing which policy should be executed by the CR at each interval I to maximize cache efficiency over a time horizon T.

Since we have continuous decision tasks during caching operations, the problem can be represented as an online learning problem. The cache has to learn which policy is more suitable to execute at each iteration. In our model, the cache operates only with one cache replacement policy at a time. Thus, the cache can measure the CE associated only with the running policy.

IV. A CACHING META-POLICY STRATEGY

Caching replacement policies behave differently according to their heuristic, but regardless of the policy, sequential $CE(\omega_i)$ measures could be seen as a sequence of random variables shaped by the running caching policy ω_i. CE measures are expected to be random since the sequence of future content requests is unknown. Also, the calculation of cache efficiency over a current interval does not consider efficiency values calculated over past intervals. This way, the measures are distributed random variables in the range [0,1].

Notice the $CE(\omega_i)$ measures of each policy may follow distinct and truncated Gaussian distributions since each policy implements particular eviction decisions. Therefore, our strategy's primary rationale is to model caching policies' efficiencies as distinct and fixed stochastic distributions. In this direction, we modeled the policy choice as an exploration and exploitation problem, typically covered by Online Learning with Partial Feedback (OLPF) algorithms.

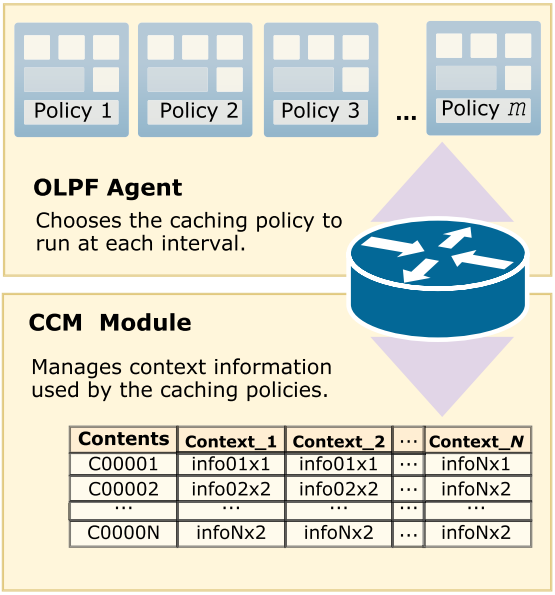


FIGURE 1. Meta-caching policy approach based on online learning with partial feedback and content/context management module. Each policy can work with a different number of *N* context information.

The second rationale of our strategy is the decoupling of the content eviction strategy from managing the context information used by the policies. For that, we consider a Content and Context Management (CCM) module able to manage the cached contents and the context information associated with the set of content replacement policies Ω available in the cache.

Algorithm 1 Caching Meta-Policy Protocol

foreach *iteration* I ***in*** *a time horizon* T **do**
  Choose policy $\omega_I \in \Omega$ according to the OLPF algorithm
  Call CCMI, ω_I, Ω to configure/operate the cache and compute the cache efficiency
  Receive the cache efficiency $CE(\omega_I)_I \in \mathbb{R}$
  Update policy choice parameters according to the OLPF algorithm
end

We combined the two procedures to propose a caching meta-policy strategy capable of learning the suitable policies online and dynamically adapting to context variations that leads to changes in which policy is best. Figure 1 illustrates the two main components: an OLPF agent and a CCM module. The cache node works with the two components by implementing the caching meta-policy protocol described in Algorithm 1. In summary, the OLPF agent is responsible for choosing the policies to run in each interval, while the CCM module operates the cache and measures the cache efficiency CE. Then, the OLPF algorithm receives the CE measured to update its parameters used in the learning process.

That model allows the cache node to act as an agent learning traditional stochastic Gaussian distributions regardless of the policy set. The strategy enhances the cache with an upper abstraction level over the content eviction strategies, and adjusts the cache to network changes that influence the policies' performance. We detail the components in the following subsections.

Algorithm 2 CCM - Content and Context Management for the Meta-Policy Strategy

Input: I, $\omega_I \in \Omega$, Ω
Output: $CE(\omega_I)_I$

Initialize cache miss and hit counters: $M = 0, H = 0$
foreach *request for content* c *during the iteration* I **do**
  if c *is not in the cache* **then**
    Increment cache miss counter: $M = M + 1$
    Elect content c' to evict from the cache according to ω_I eviction logic
    Add new content c in the cache
    foreach $\omega_i \in \Omega$ **do**
      Remove context data used by ω_i and related to c'
    end
  else
    Increment cache hit counter: $H = H + 1$
  end
  foreach $\omega_i \in \Omega$ **do**
    Update context data used by ω_i and related to c
  end
end
Compute and return $CE(\omega_I)_I = \frac{H}{M+H}$

A. ONLINE LEARNING AGENT WITH PARTIAL FEEDBACK

OLPF describes a set of sequential decision-making problems. An agent interacts with the environment online by deciding to execute one action over a finite set of actions. The chosen action yields a numerical reward after being executed, and the reward of the unplayed actions remains unknown. The agent has no previous knowledge about the mechanism generating the sequence of rewards for each action. This way, there is always uncertainty about whether the agent made the best choice. The agent can explore a new action at each iteration or exploit the best action learned in previous iterations. The agent has to explore to learn its possibilities and also exploit to increase its gains in the long term.

In the caching meta-policy strategy, the decision-maker agent is represented by the cache node, and each cache replacement policy is an option to be chosen. For in-network architectures, cache nodes are the network routers enhanced with storage capacity. The cache works with a fixed set of policies. Then, in a continuous task, the agent chooses a policy to run inside a predefined interval of time. At the end of each interval, the agent receives the reward associated with the policy and evaluates whether the same caching policy will be running in the next interval or another policy. The reward

is the cache efficiency measured inside the interval and relies only on the running policy. This way, the OLPF agent uses the reward obtained in each interval to learn the policies' distributions, i.e., the cache efficiencies' distributions. The agent is agnostic to the caching policy eviction logic and does not associate the choice with any traffic pattern or network-related characteristic.

Caching policies' performance may vary according to traffic characteristics changes, especially for highly dynamic networks with intermittent wireless communication. In that context, the distributions are expected to be non-stationary. Models with stationary distributions have no variation in which option will achieve the highest cumulative rewards over time. This way, the learning process focuses on finding which one is the best option. However, in non-stationary stochastic problems, the distributions may infrequently change in a time horizon. The learning process, thus, requires different strategies to adapt to possible variations of best arms. Therefore, the agent may employ algorithms able to deal with non-stationarity in stochastic models.

The stationarity degree and characteristics depend on the set of policies. The strategy effectiveness and adaptability rely on the employment of OLPF algorithms appropriate to the distributions' profile of changes.

B. CONTENT AND CONTEXT MANAGEMENT MODULE

Since the caching meta-policy strategy consults the OLPF agent at each iteration I to configure the cache with the policy chosen by the agent (see Algorithm 1), each iteration can be executed with a different policy ω. A requirement to allow the dynamic change of policies during the cache operation is to maintain the context information associated with each policy's eviction logic in the set Ω. This way, the strategy implements a CCM module to store the context for all contents in the cache. The management module keeps the context information used by all the policies in the cache policy set Ω regardless of which policy is executing. Besides, the module manages the content eviction engine by matching the stored context information with the context information used by the running replacement policy.

Upon the beginning of an iteration, a chosen policy begins its execution relying on the stored context. This way, it is possible to continue the cache operation from the current cache state left by the previous running policy. Otherwise, it would not be possible to change policies online. Algorithm 2 presents the pseudo-code of the CCM module.

The policy set can vary in number and replacement logic exploring different context aspects, such as content, router, and network properties. Several context factors can influence the performance of policies. Therefore, there is no single criterion for choosing the candidate caching policies to compose the set. Each cache on the network can work with a different set. The meta-policy strategy is a mechanism of choice, and the learning will converge to the most suitable policy among the options in the chosen set. In other words, the cache performance will converge to the performance obtained if only the best policy present in the set is executed.

V. EXPERIMENTAL EVALUATION

To demonstrate the applicability and benefits of using the caching meta-policy strategy for choosing suitable caching policies, we have carried out a simulation-based study through the NDN architecture. NDN in-path cache works as an opportunistic cache to distribute contents across the network. This section details the evaluation methodology and discusses the experimentation settings.

A. EVALUATION ENVIRONMENT

The generic aspects of the caching meta-policy encompass both the policy set with its corresponding context information and the OLPF algorithm (Figure 2). The strategy allows the employment of different policy sets and OLPF algorithms for each cache. It is also generic regarding the network type in which the cache operates. This section details the evaluation methodology we carried out as a proof of concept of the strategy. We present the settings of simulation-based experiments in caching routers implementing the NDN architecture, with common caching policies and basic OLPF algorithms. We expect the performance of a cache that executes the meta-policy to converge to the performance of the policy best suited to the context of the cache's operation.

We have implemented the meta-caching strategy in a modified version of the ndnSIM simulator [22], [23] coupled with the ns3-gym framework [24]. The ndnSIM is an open-source NDN simulator to reproduce discrete-event network scenarios, and the ns3-gym framework is designed to support the interaction of machine learning agents with the network environment. This way, a learning agent based on the ns3-gym framework can interact with the NDN cache node created by the ndnSIM. To evaluate our proposed strategy, we have implemented cache replacement policies on the ndnSIM and online learning algorithms on the ns3-gym. In the following subsections, we describe the policies and algorithms and detail the scenario settings.

B. CACHE REPLACEMENT POLICIES

For our proof-of-concept evaluation, the caching policy set contains four basic policies: LRU, LFU, FIFO, and Random. As described previously in section II, LRU removes the last accessed content, LFU removes the last frequently used content, FIFO removes the oldest content, and Random removes one content randomly. Those are traditional replacement policies inherited from the memory management of operating systems and used in web cache networks.

To exemplify possible efficiency distributions for the distinct caching policies, Figure 3 illustrates the execution of LFU, LRU, FIFO, and Random, in a single cache scenario running individual policies separately. The scenario had a fixed catalog of 20.000 contents, static content popularity following the Zipf model with $\alpha = 0.8$, relative cache size of 5%, and request rate of 10 requests/s. We illustrate each

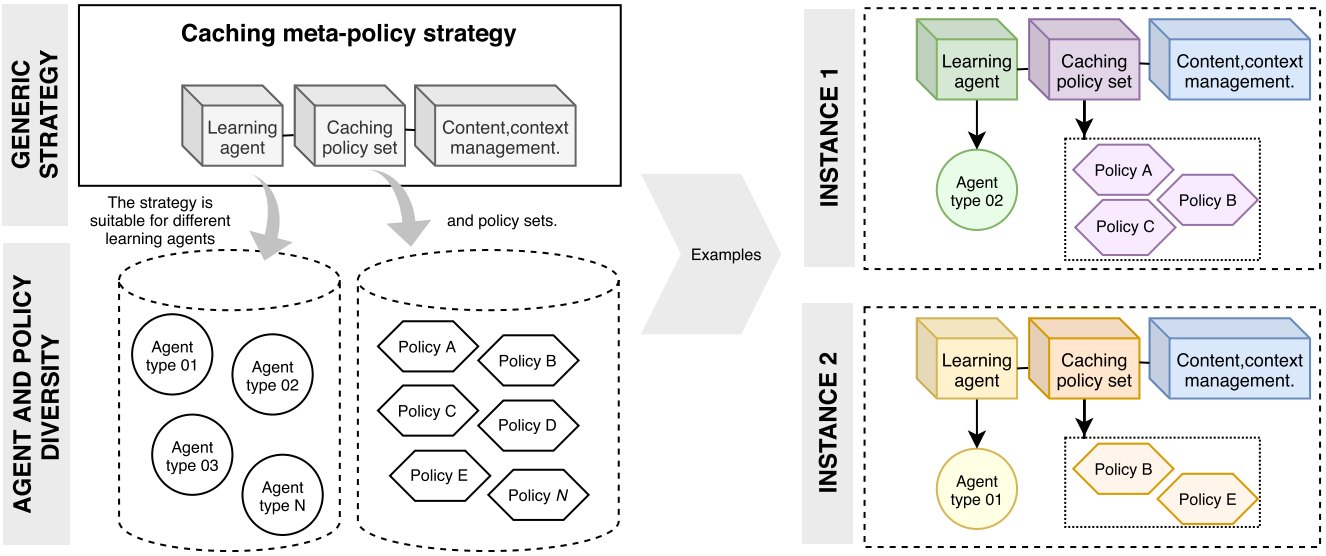


FIGURE 2. The meta-policy strategy is generic and can be instantiated with different policy sets and learning agents. The agents on the left side represent possible bandit algorithms that could be instantiated. Each CR in an ICN can have one instance of a chosen agent. Thus, INSTANCE 1 and INSTANCE 2 (right side) would represent two different CRs, each running its caching meta-policy strategy with its chosen bandit algorithm.

policy's cache efficiency in a day, measured in 5 seconds intervals. The figure shows the distribution of policies' performances from three different perspectives: (a) the log-scale graph of all immediate measures by interval, (b) the total cache efficiency over time, and (c) the normal distribution for each policy. Under the Independent Reference Model (IRM), LFU has optimal performance but requires some time to populate the cache with the most popular contents. Notice the variance in the first iterations, in which other policies performed better than LFU. Upon increasing the iterations number, LFU started to perform better, and the distributions became stationary. In this scenario, FIFO and Random obtained similar distributions, almost indistinguishable in the graph.

Different scenario settings would generate distinct distributions for the same policies. In the same way, different policies would generate new distinct distributions. Our proposed strategy stems from the analysis of similar distributions to learn which ones should be used by the cache.

The caching meta-policy strategy has native context adaptability associated with the policy set diversity. Therefore, we have implemented the ARC policy to compare the ARC adaptive behavior to the behavior of our strategy. Moreover, we extended the experiments to assess the generality of our strategy by including ARC in the policy set used by the learning agents. For both purposes, we made different combinations of policies to compose the set used by the learning agents. The ARC policy uses a learning rule to dynamically adjust the cache by balancing content recency and frequency aspects. We discuss more details of the policy and the results of experiments in section VI-D.

In the following, we present the online learning algorithms we have employed in the learning process of the caching meta-policy strategy.

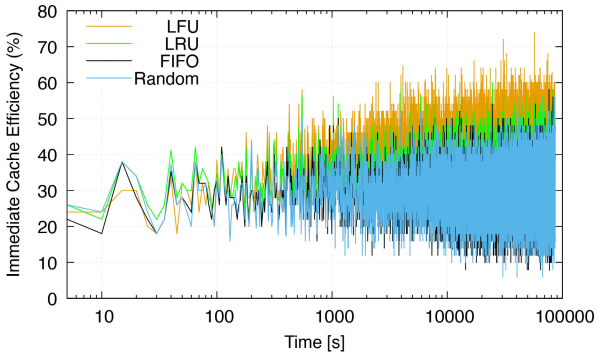
C. OLPF ALGORITHMS

We evaluated our strategy with Upper Confidence Bound (UCB) algorithms as learning agents. UCB is a Multi-armed bandit (MAB) strategy successfully used to solve stochastic bandit problems. In the stochastic setting, bandit algorithms usually estimate the arm's values by incrementally averaging its rewards in a time horizon. The more an action is taken, the more confidence we have that the average will reflect the action's actual value. UCB strategies add a fixed confidence interval to each arm's mean to estimate the expected arm's values optimistically. It is based on the concept of optimism in the face of uncertainty about the mean values. The strategy can gradually reduce the interval as the bandit gains more confidence in the mean values.

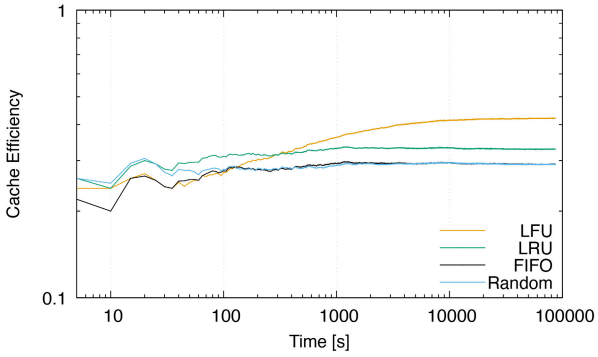
The literature presents several variations of UCB algorithms. A standard UCB deals with stationary problems in which the average considers the entire distribution evenly. In non-stationary scenarios, it is possible to include a discount factor to give higher weights to more recent rewards, similar to the ones proposed by [25] and [26]. Another alternative is to consider only the most recent measurements in a sliding window over time [25].

The experiments considered UCB algorithms for non-stationary stochastic bandits that choose the policy ω for the interval I according to the following equation:

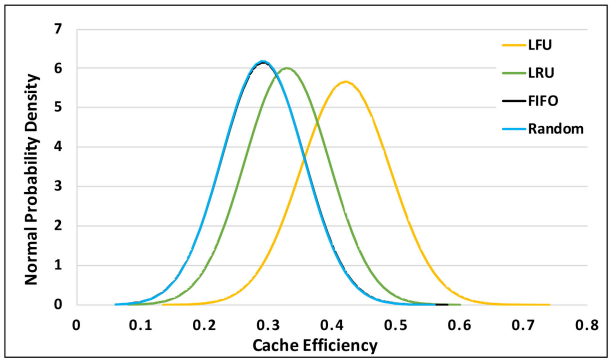
$$\omega_I = \underset{\omega \in \Omega}{\operatorname{argmax}} \left[\overline{CE}_I(\omega) + \gamma \sqrt{\frac{\ln I}{N_I(\omega)}} \right] \tag{3}$$



(a) Immediate efficiency measures at each interval of time, according to equation 1.



(b) Total cache efficiency over time, i.e., at each iteration, we measured the total content requests and total cache hit from the simulation's start to the corresponding iteration.



(c) Graph of probability density function for the distributions obtained by each policy.

FIGURE 3. Distribution of policies's performance for a single cache under IRM model.

in which $\overline{CE}_I(\omega_t)$ is the average efficiency for policy ω obtained before iteration I, and the square root part is the confidence interval; $N_I(\omega)$ is the number of times policy ω has been chosen, and γ is a fixed parameter to tune the effect of the confidence interval thereby controlling the degree of exploration [26].

The algorithms are (i) UCB with discount factors (UCB$_d$) and (ii) sliding-window UCB (SW-UCB). For the UCB$_d$, we have applied the incremental exponential recency-weighted average (ERWA), according to equation 4,

$$\overline{CE}_{I+1}(\omega) = \overline{CE}_I(\omega) + d\Big[CE_I(\omega) - \overline{CE}_I(\omega)\Big] \quad (4)$$

in which $d \in (0, 1]$ is a step-size parameter that works as a discount factor in the average learning process. The discount factor adjusts distinct weights over the reward distribution, wherein higher values emphasize recent rewards. So, in the experiments, we adopted $d = 0.2$ and $d = 0.8$ as two opposite values to evaluate the bandit's adaptability. The SW-UCB used the simple average and tuned the window size parameter according to [25].

D. SCENARIOS AND EVALUATION METRICS

We have conducted experiments to analyze our strategy effectiveness in converging to the best policy for different scenarios. The scenarios contain variations on content request patterns, cache sizes, and locations of the cache node into the topology. We also analyzed the impact of different interval times that the learning agent interacts with the cache. The evaluation metric for all scenarios is the cache efficiency obtained by applying the meta-caching strategy. The strategy efficiency was compared with the correspondent benchmark scenario consisting of one replacement policy throughout the simulation period. Table 2 summarizes the scenarios parameters and their respective values.

1) NETWORK TOPOLOGY

Regarding the network topology, we divided the evaluations into single-cache and multi-cache scenarios (Fig. 4). In the single-cache, we have tested the impact of different access patterns, cache sizes, and agent iteration times.

In the multi-cache scenario, we present an analysis regarding the impact of different node positions on the network. We aim to explore the variance of suitable policies for the individual caches. To this end, we first carried out experiments with a tree topology composed of one intermediate caching node and three edges caching nodes. Then, we expand the study on different intermediate positions with nine caching nodes arranged in a 3 × 3 grid topology. We placed the producer and consumers at opposite ends of the same grid's diagonal. The consumer issues content interests that transverse the caching nodes up to match searched contents in respective caches or the content producer. When a cache node does not have a requested content, it broadcasts the incoming content interest to neighbor nodes. This setting allows us to simulate dense cache connections while exploring the different in-network cache positions. The different positions allow each cache to have unique traffic views and thereby possible variations of suitable policies. All caches have similar sizes.

2) DATASETS

To accomplish variations of the access pattern, we have performed experiments with the IRM request model implemented in the simulator, and also on three public datasets

TABLE 2. Scenarios parameters.

Parameter	Values			
Caching policy set	{LRU, LFU, FIFO, Random}, {LRU, LFU, FIFO, Random, ARC}, {LRU, LFU, ARC}, {LRU, LFU}			
Placement policy	Leave Copy Everywhere (LCE)			
OLPF algorithms	$\text{UCB}_{d=2}$, $\text{UCB}_{d=8}$, SW-UCB			
Agent iteration interval	2, 3, 4, 5, 10, 15, 20, 25, 30 s.			
Number of cache nodes	1, 4, 9			
Node positions	edge, intermediate positions			
Cache size	2, 4, 5, 6, 8, 10, 12, 14, 16, 18, 20%			
Content request pattern	**IRM model**	**Boston sample trace (1 day)**	**Youtube sample trace (1 day)**	**DEC sample trace (1 day)**
Content library size	1.000, 20.000	≈ 20.000	≈ 30.000	≈ 600.000
Content Zipf(a)	0.8	≈ 0.80	≈ 0.53	≈ 0.63
Total content request	5, 10 /s.	≈ 10.000	≈ 50.000	≈ 1.5 million

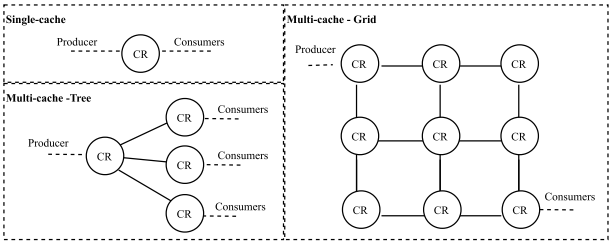


FIGURE 4. Cache network topologies for the experiments. The network devices implement the NDN protocol stack, and all the NDN routers have cache capacity.

suitable for caching experimentation: a dataset of user requests for Youtube videos [27], a Web Proxy dataset from Digital Equipment Corporation (DEC) [28], and web traffic traces from Boston University [29]. All datasets contain trace files with users' content requests and the respective timestamps. Figure 5 illustrates sample traces of each dataset. The figure shows the different content request rates over time and the distribution of content popularity in a sample day. We carried out experiments with different periods on the datasets and selected one-day traces to present the results.

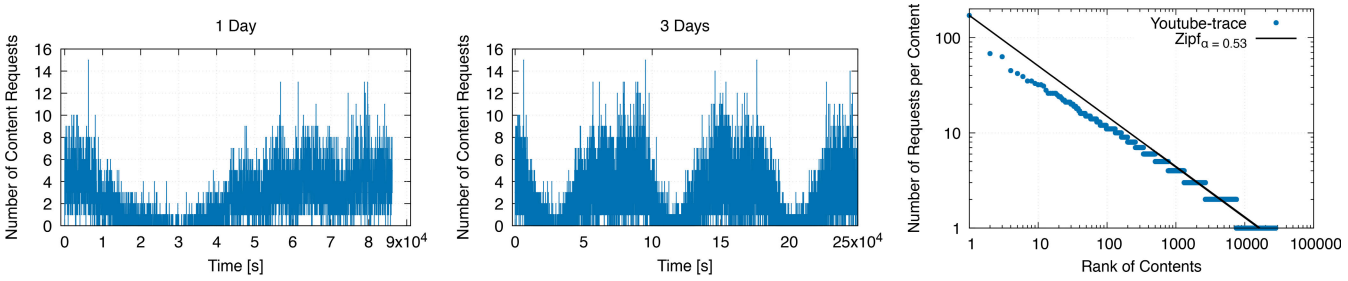
VI. RESULTS

This section presents our findings in applying the meta-policy strategy under different network scenarios. In general, the policies presented distinct behaviors for each scenario, and most strategy instances performed close to the best-fixed policy for all scenarios.

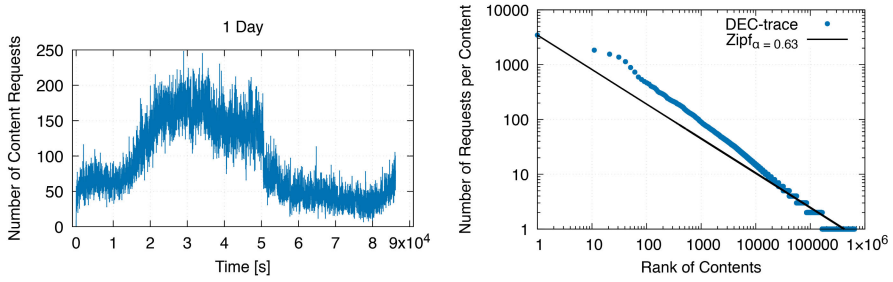
Figure 6 depicts the results of applying the caching meta-policy strategy in scenarios with different content request patterns. The figure shows the total cache efficiency over time, i.e., at each iteration, we measured the total content requests and total cache hit from the simulation's start to the corresponding iteration. We first elaborate on the meta-strategy application in the single-cache IRM scenario of section V-B with the three variations of bandit algorithms. The agents explored all policies in the set according to its learning algorithm. As we have described, LFU is the best policy in the set for the IRM model, followed by LRU. The results show that both $\text{UCB}_{d=0.2}$ and $\text{UCB}_{d=0.8}$ performed closer to LFU in the long run. Even applying discount factors, UCB_d is influenced by the initial distribution values. However, in this scenario, both lightweight and aggressive discount factors could minimize the influence of lower LFU initial values. A stationary strategy would not be able to have that effect. Although SW-UCB considers more recent distribution values and is not impacted by the initial values, the bandit had struggled to adapt by showing a slower learning curve. One possible reason is that the policies may obtain immediate measures with nearby values that may overlap in some iterations. Hence the importance of learning with many iterations of the agent to estimate more confident policy values. Figure 7 complements the IRM results by showing (a) the immediate measures of cache efficiency at each 5 seconds intervals, and (b) the distribution of policies choices by bandit algorithms.

It is worth mentioning that the distributions perceived by the bandit in the learning process are not exactly the same as those obtained when a single policy is executed throughout the entire cache execution. Inside the bandit, the policies obtain different reward values due to the different cache states at each iteration that starts with a new policy. Even with fragmented executions, the policies manage to maintain their distribution characteristics. However, we notice that in such a static setting, frequency-based policies may perform optimally when executed alone but may not perform the same way inside the bandit. The reason is the possible loss of high-frequency contents during other policies' execution. In similar cases, the bandits can only approximate their performance.

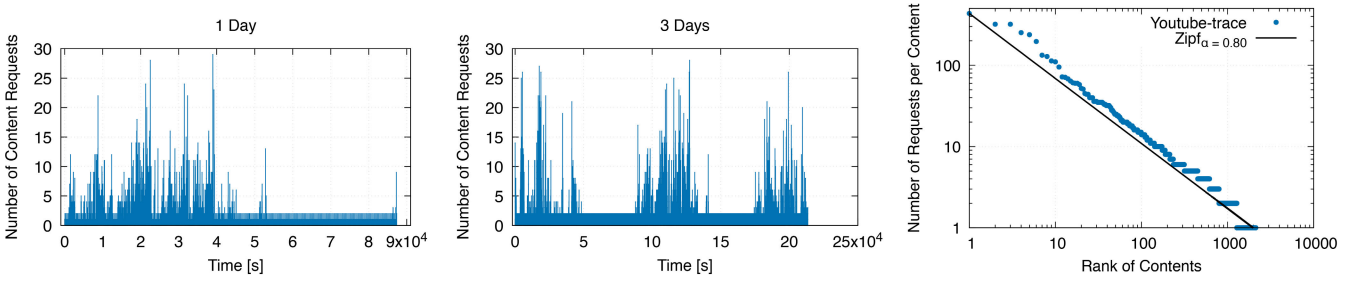
Figure 6 also depicts the cache efficiency for one sample day of (a) Youtube, (b) DEC, and (c) Boston traces. Unlike the IRM model, the real web traces present degrees of temporal locality between content requests. Temporal locality



(a) Youtube.



(b) DEC.



(c) Boston.

FIGURE 5. Sample traces for three distinct real content request datasets. The number of content requests were measured in 5 seconds intervals. The distributions of content requests per content refers to measures in one sample day.

describes correlation properties in content requests, in which recently accessed contents are likely to be reaccessed shortly. Policies such as LRU tend to present better performances when the temporal correlation is the prevailing characteristic of the content request pattern.

For Youtube trace, LRU stands as the best policy in the set, and the performance of all bandits approached LRU. The other policies performed well and approximated to LRU, except for LFU, which had degraded performance as the simulation progressed. Therefore, in more dynamic scenarios such as with content popularities changes, the bandits can perform as well as the best-fixed policy in the long run. In such a scenario, even stationary bandits would perform well.

The DEC trace results presented interesting behavior with the variation of the best policy during the simulation time. After an initial period in which all policies appeared to perform almost evenly, LFU started to perform better and became the best option. Still, LFU gradually lost performance due to content popularity profile changes, and LRU remained stable as the best choice. Since LRU uses the least recent approach to evict content, it is more appropriate for dynamic scenarios. However, the cache size is also a factor that impacts cache efficiency. The increase in cache size could mitigate LFU performance degradation in such cases. Regarding the bandit's behaviors, $\text{UCB}_{d=0.8}$ demonstrated better adaptability. The bandit approached LFU and then adapted to LRU when LRU became the best option. $\text{UCB}_{d=0.2}$ got stuck in LRU from the beginning, probably due to its higher initial distribution values when LFU went through its natural learning curve. The trace results reaffirmed the better adaptability of the meta-policy strategy to more dynamic scenarios since most bandits performed as well as the best-fixed policy in the long run.

For the Boston trace, the policies behaved very similarly to the Youtube trace; however, the agents could not approach the LRU performance in the same way. The reason was the absence of request patterns in approximately half of the trace time. In the trace, requests are practically extinguished in the second half of the day. Thus, the performance measurements

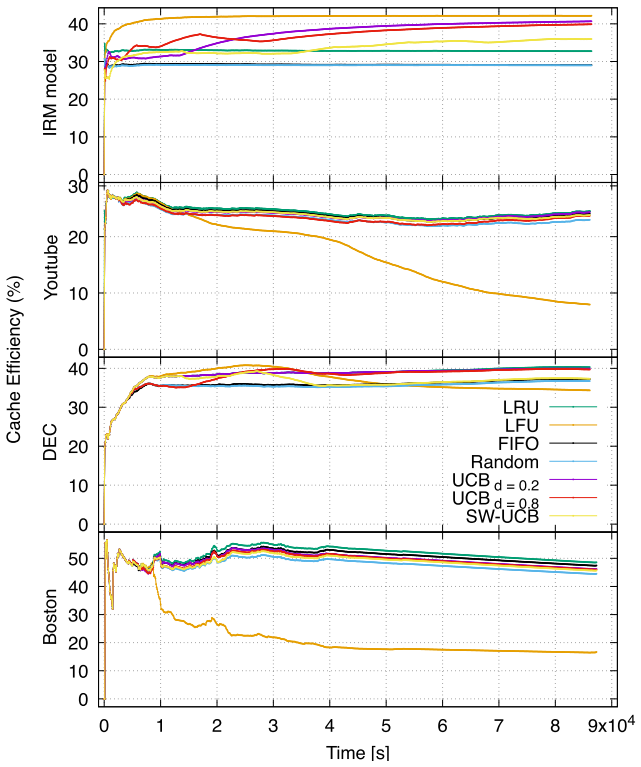
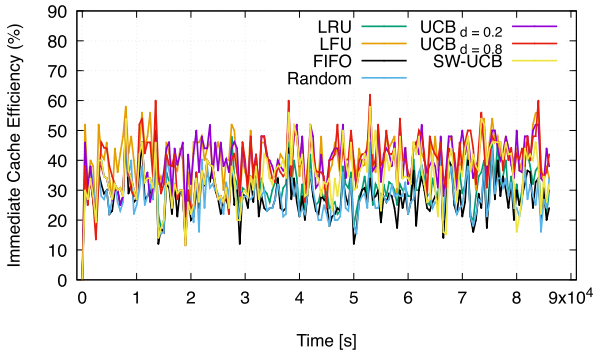
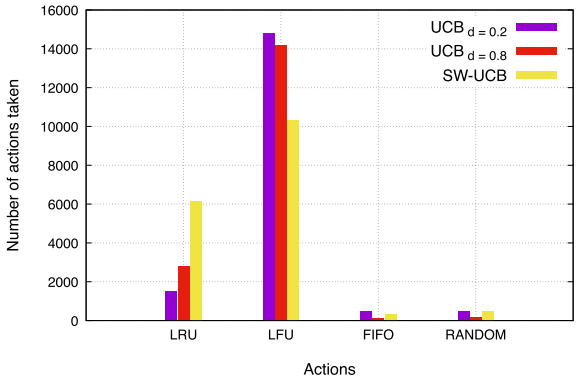


FIGURE 6. **Cache efficiency for single-cache scenarios with different content request patterns. Simulation parameters: relative cache size: 5%, iteration time: 5s, $\gamma = 0.2$, warm-up period: 200 s.**



(a) Immediate efficiency measures at each interval. The graph show one every 100 measures to improve visualization.



(b) Distribution of actions taken by OLPF agent.

FIGURE 7. **Cache efficiency for the single-cache IRM scenario. Simulation parameters: relative cache size: 5%, iteration time: 5s, $c = 0.2$, warm-up period: 200 s; 20.000 distinct contents, Zipf(a)=0.8, request rate: 10 requests/s.**

obtained by the agents in the absence of requests were not able to measure the value of each policy. Still, the learning process in the first half of the time allowed the agent to perform close to the best policies in the set.

A. IMPACT OF AGENT ITERATION TIME INTERVAL

The iteration interval determines the time an agent will intervene in the cache to evaluate the running policy and perform policy changes. As such, the interval sets the time a policy has to run before being evaluated. The shorter the interval, the more interactions the agent will make in a time horizon, and thereby the agent will have more confidence in the estimated policies' values. However, small intervals may not adequately reflect the value of the evaluated policies, as the policies need space to show their ability to assist content requests. In contrast, long intervals lead to slow convergence in agent learning.

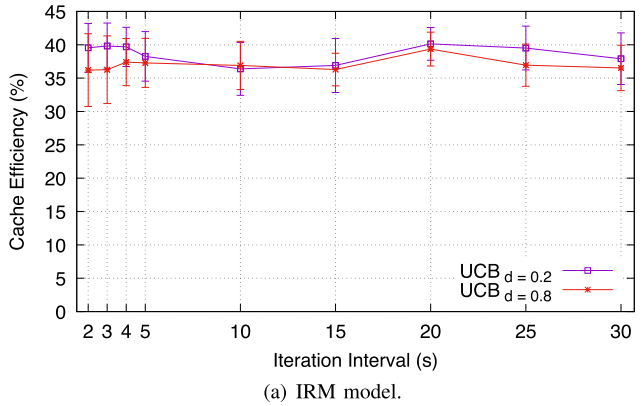
We assessed the impact of different time intervals on cache performance when applying our meta-policy strategy. Figure 8 depicts the results of experiments with the IRM model and Youtube traces. There were variations for each interval as the distributions obtained are distinct, but the average performance for each interval size remained nearby to each other. The variation was more significant for the IRM experiments, while the cache performance remained almost constant for the Youtube experiments.

In general, the cache maintained the performance regardless of the time interval used by the agent. The experiments used the static time configuration, but the cache can adopt dynamic approaches to increase or reduce the interval according to network variations.

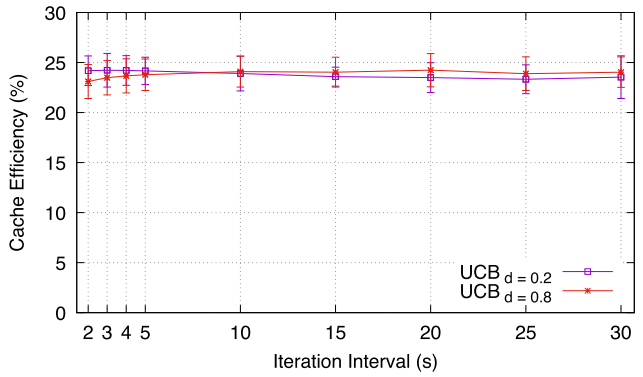
B. IMPACT OF CACHE SIZE

The cache size is one of the context factors correlated to the policies' performances and can influence the policy choice [8]. We carried out experiments with cache size variations in scenarios with the IRM model and Youtube traces to analyze the impact on our proposed method. Figure 9 depicts the results. In general, the direct impact of cache size variations relied on the policies' performances. However, we notice a not negligible impact on the average performance of the agents.

Regarding the policies performances, the increase in cache size usually causes two effects: first, the cache performance naturally increases regardless of the policy since there is more space to store popular content, but the performance gain is not linear with increasing cache size; and second, the policies performances tend to converge for reasonably large cache size. The IRM and Youtube scenarios presented both effects but with very different granularities, as shown in the picture.

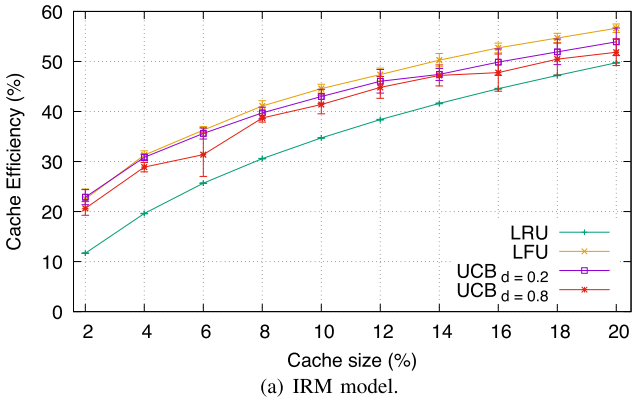


(a) IRM model.

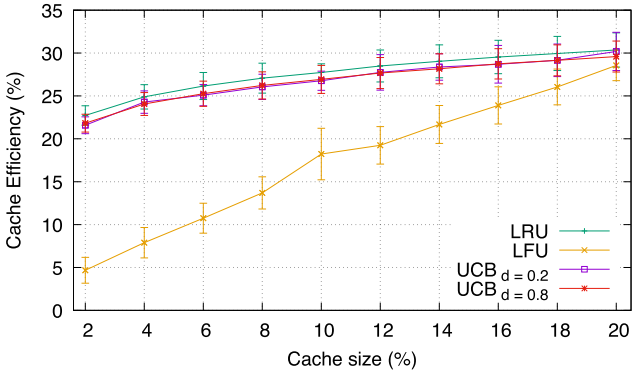


(b) Youtube trace.

FIGURE 8. Average cache efficiencies for different agent iteration intervals in a single-cache scenario. Average over 10 runs. Common simulation parameters: duration of each run: 1 day, relative cache size: 5%, $\gamma = 0.2$, warm-up period: 200 s; For IRM model: 20.000 distinct contents, Zipf(a)=0.8, request rate: 10 requests/s.; The Youtube trace samples are similar to the sample in Table 2.



(a) IRM model.



(b) Youtube trace.

FIGURE 9. Average cache efficiencies for different cache sizes in a single-cache scenario. Average over 5 runs. Common simulation parameters: duration of each run: 1 day, iteration time: 5s, $\gamma = 0.2$, warm-up period: 200 s; For IRM model: 1.000 distinct contents, Zipf(a)=0.8, request rate: 5 requests/s.; The Youtube trace samples are similar to the sample in Table 2.

Regarding the caching meta-policy strategy, the agents choose between the four replacement policies as in the previous experiments. The figure shows the comparison with the two most representative policies. In the IRM experiments, the agents generally achieved better average performance for smaller cache sizes. There were more variations for larger cache sizes, as seen from the increase in the standard deviation. That means the UCB agents explored the policies more. This behavior could be associated with the slight convergence of policies' performance and the learning pattern of UCB algorithms. For the Youtube experiments, the agents performed very similarly for all cache sizes. The combination of both results reinforces that variations in the policy set's performance pattern can influence the agent's learning rate, not the cache size directly.

C. IMPACT OF NODE LOCATION IN THE NETWORK

This section presents our experiment in the multi-cache topologies described in section V-D.1. The experiments aimed to explore the variance of suitable policies for different cache node positions on the network, and thereby our strategy's effectiveness in learning accordingly.

Multiple cache levels naturally present variations in the traffic characteristics perceived by each cache. The reason is the filtering effect when a cache closer to the user hits a content request. The cache does not propagate that request to the rest of the network and propagates only the miss requests to upper-level caches. This behavior modifies the original characteristics of the traffic and directly impacts each cache's choice of policies. Therefore, a homogeneous policy configuration may not adequately address the individual cache needs. Besides, a policy running in one cache can influence the efficiency of all neighbor caches. Therefore, an appropriate choice of policies would consider that interaction to ensure overall network efficiency. Yet, this section presents a simple model of independent and distributed bandits. We show that, even without collaboration, most bandits can learn and adapt the policies with all routers executing the meta-policy selfishly. We compare the results with the homogeneous policy setting, i.e., all routers with the same replacement policy.

We first show the results of the experiment in the tree topology. The topology has one intermediate router and three access routers. We aimed to evaluate the meta-policy adaptability in the intermediate node while all edge routers are also running the meta-policy. Each access router had distinct

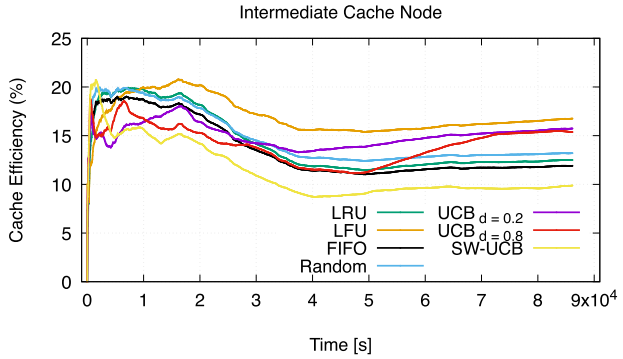


FIGURE 10. Cache efficiency for the intermediate node position in the tree topology. Simulation parameters: relative cache size: 2%, iteration time: 5s, $\gamma = 0.2$, warm-up period: 200 s; For IRM model: 20.000 distinct contents, Zipf(a)=0.8, request rate: 2 requests/s; For Dec trace: ≈ 65.000 requests, ≈ 37.000 contents; For Youtube trace: ≈ 45.000 requests, ≈ 30.000 contents.

traffic profiles: IRM traffic, parts of DEC trace, and Youtube trace. Thus, the intermediate node receives filtered and mixed traffic of all three edge nodes. Figure 10 depicts the results for the intermediate node. The edge nodes maintained the cache behavior discussed before in the single cache experiments according to their respective traffics. Meanwhile, the intermediate cache node was also able to adapt its policy on-demand. The graph shows the efficiency of an agent in the intermediate node relative to the efficiency of all other edge nodes' efficiencies running the same agent type. Likewise, the single policies' efficiencies are relative to the same policy running in all edges.

To further explore the adaptability in intermediate cache nodes, we performed experiments in the 3×3 grid topology presented in section V-D.1. We placed the content consumers at one cache node to isolate the edge effect, and then we evaluated the agent's ability to learn from the different intermediate node positions. Figure 11 illustrates the caching nodes' efficiencies for the experiment with IRM traffic. We kept the same simulation parameters as the single-cache IRM scenario presented earlier (see Figure 6), e.g., the same consumer request rate and the number of distinct contents. Notice the variance in both cache efficiencies and policy behaviors according to the node position. In the present scenario with equal caches setting, the variations are mainly due to each node's particular traffic view. The caches enhanced with the meta-policy were able to adapt their policies on-demand. The bandits presented variations in efficiency for each node position as the single policies due to the different traffic views in each position. However, the bandit configurations improved the overall network efficiency over the homogeneous policy settings. $\text{UCB}_{d=0.2}$ stood out with the highest performance levels for most node positions.

As mentioned earlier, the bandits' performances are linked to the policy set. FIFO has performed poorly in the cache network scenario for most node positions, and the bandits managed to maintain good performance even using FIFO in their continuous learning process. Regarding the cache efficiencies compared with the single-cache IRM scenario, as expected, the overall cache network efficiency improves, i.e., the sum of all individual cache efficiencies, since we have increased the total system cache capacity. Naturally, the average cache efficiency reduces due to a combination of the hierarchical grid structure and the configuration parameters of the simulated scenario (e.g., content popularity, request rate, and cache size).

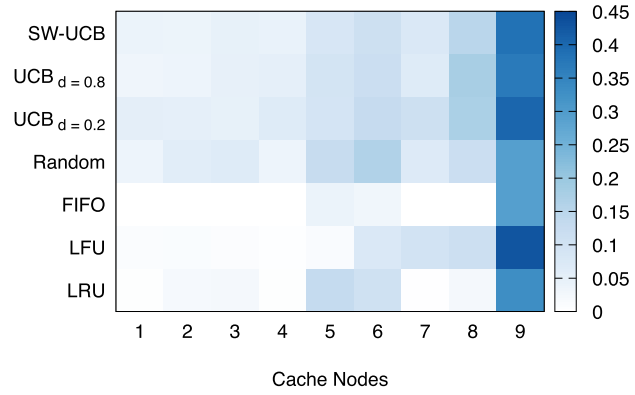
In caching networks, each cache could work with different policy sets and bandit algorithms. More realistic traffic presents variations in the temporal locality patterns perceived by each cache. The traffic received by caches closer to the users presents strong temporal localities. As cache levels filter requests, the temporal locality intensity becomes gradually weakening, and the traffic profile at upper-level caches becomes more random. Real caching networks would similarly benefit from the online policy adaptation.

D. EXPERIMENTS WITH ADAPTIVE REPLACEMENT POLICY

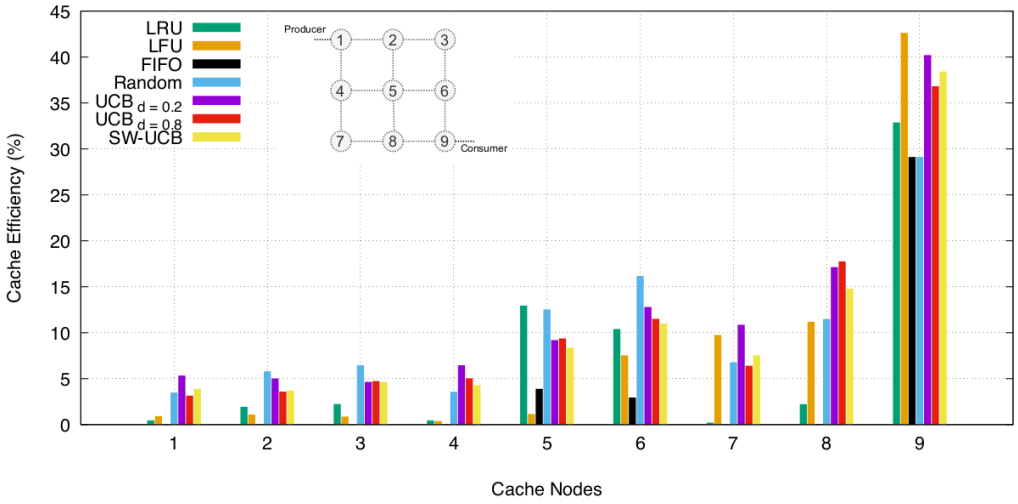
This section presents the experiments we have performed with the adaptive policy ARC. We ran the policy alone to see its adaptability in our scenarios, and we included ARC in the policy set to assess the generality and inclusiveness of our strategy.

The ARC is an adaptive policy able to perform adaptations online when the traffic pattern changes between recency and frequency aspects. The policy works with two LRU lists (L_1 and L_2), one to capture recent one-timer content requests and the other to capture content accessed more than once, capturing the frequency aspect. The total size $S = |L_1| + |L_2|$ of the two lists is fixed, but their individual sizes vary according to the traffic pattern. To carry out the adaptation, ARC applies a learning rule of constantly resizing the individual LRU lists based on the eviction history. In addition, the policy maintains a list of content eviction history of size S. This way, the context information used by ARC and saved by the CCM module of our strategy are: the content frequency and recency (same as LFU and LRU), the size of the lists, and the cache eviction history.

Figure 12 depicts the results. The figure shows experiments with three traffic patterns: the IRM model, Youtube, and DEC traces. We plotted the caching performance when running the single policies used in the previous experiments (LRU, LFU, FIFO, Random) and ARC. Also, the figure shows the caching performance when applying the meta-policy with different policy set configurations. We have varied the number of policies in the set. That means varying the number of arms used by the learning agent. The figure shows *a*) five-armed UCB agents, running with the policies LRU, LFU, FIFO, Random, and ARC; *b*) three-armed agents, with policies LRU, LFU, and ARC; and *c*) two-armed agents, with policies LRU and LFU.



(a) Heatmap of individual cache efficiencies.



(b) Histogram of individual cache efficiencies.

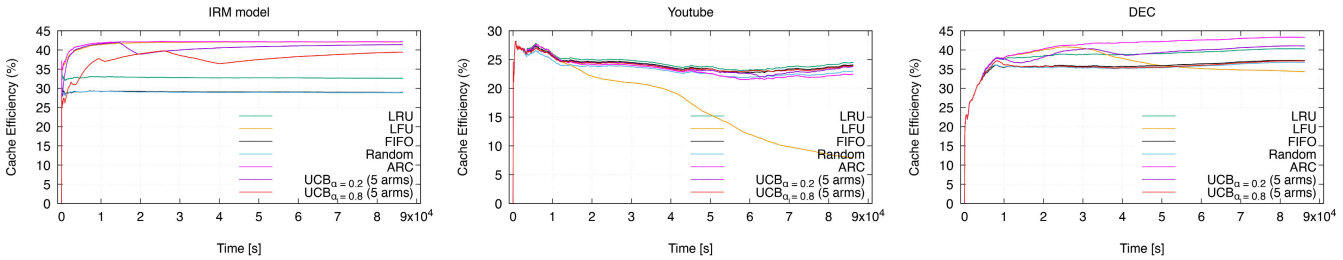
FIGURE 11. Cache efficiencies for different node positions in the IRM 3 × 3 grid scenario. Simulation parameters: relative cache size: 5% for each cache node, iteration time: 5s, $\gamma = 0.2$, warm-up period: 200s, distinct contents: 20.000, Zipf(a)=0.8, request rate: 10 requests/s.

In our experiments, the ARC policy has shown good adaptation to the different traffic patterns. Still, its position in the rank of best policies varies: in the IRM scenario, ARC performed as well as LFU in the long run, presenting a better performance at the beginning of the simulation; with the Youtube trace, ARC presented a lower performance than all the other basic policies; meanwhile, in the DEC scenario, ARC increased the cache performance and stood out as the best policy. The experiments ratify that ARC alone may not suit better in all scenarios.

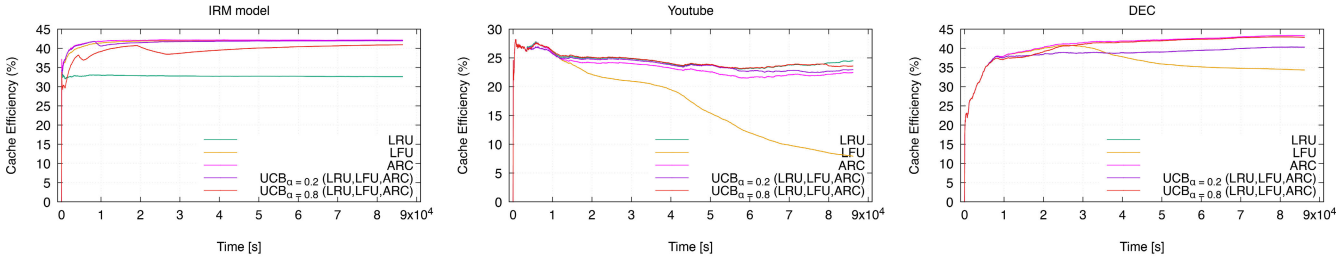
The meta-policy strategy also adapts to distinct traffic patterns, but the adaptation process is different because it continuously converges to the best policy in the operation set. This way, in our scenarios, we benefit from including ARC in the policy set used by the learning agent. Parts (*a*) and (*b*) of Figure 12 shows the experiments including ARC in the policy set. Most UCB agents generally converge to the best policy in their policy set. The convergence had different degrees depending on the number of policies in the set.

The tree-armed agents with LRU, LFU, and ARC had the best adaptation, in which the agents converged to perform the best policy in all scenarios. For the IRM scenario in which ARC presented a slightly better performance than LFU, the UCB agent with discount factor $d = 0.2$ adapted to ARC at the beginning of the simulation. It remained during almost all the simulation time. The agent has chosen ARC 93% of the time. For the Youtube trace, both UCB configurations converged to LRU as the best policy, but the UCB with $d = 0.8$ achieved better performance by maintaining the convergence for a more extended period. In the DEC trace, the UCB with $d = 0.2$ got stuck in LRU from the beginning, almost indistinctly in the plot, and the configuration with $d = 0.8$ converged to ARC as the best policy.

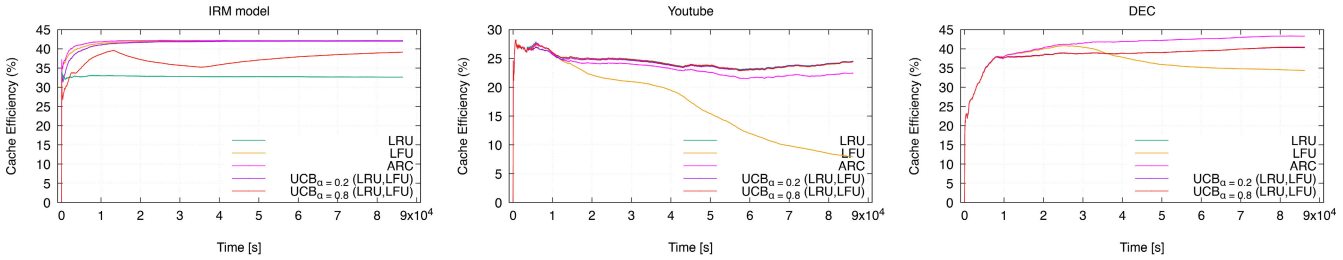
We also performed experiments with two-armed agents choosing between LRU and LFU (part (*c*) of Figure 12). That is the closest behavior to compare with the ARC policy. The agents and the single policy ARC are similar because both will balance frequency and recency aspects. However, the agents will execute one policy at a time, and ARC maintains



(a) Five-armed UCB agents, with policy set = LRU, LFU, FIFO, Random, and ARC.



(b) Three-armed UCB agents, with policy set = LRU, LFU, and ARC.



(c) Two-armed UCB agents, with policy set = LRU and LFU.

FIGURE 12. Comparisons with ARC policy in single-cache scenarios with different content request patterns. Simulation parameters: relative cache size: 5%, iteration time: 5s, $\gamma = 0.2$, warm-up period: 200 s.

both aspects in the cache simultaneously. The results show that all UCB agents performed the best policy for each scenario. We also plotted the ARC performance to emphasize that the agents' performance relies on the policy set. Therefore, the experiments assess the benefits of choosing representative policies to compose the set and the generality of the strategy by including adaptive policies.

E. REMAINING REMARKS

This work focused on evaluating the meta-policy model applicability for on-path in-network caching. To this end, we have implemented and tested standard caching policies in simulated ICN scenarios. However, there are a variety of different caching policies that include different aspects of context in their eviction logic. We believe it is crucial to evaluate the impact of the meta-policy approach on implementing other policies to support the generality aspect of the policy set.

For example, the meta-policy approach would impose a limitation on the normal function of Time-To-Live (TTL) based policies. Those policies can evict content after a specific lifetime or after it reaches a certain age in the cache. Considering the meta-policy working with a policy set containing a TTL-based policy, content that reaches its lifetime during the interval of another policy execution could not be removed from the cache. One approach would be to implement the policy differently, in which the eviction of expired content would be triggered at the beginning of the interval of the TTL-based policy and continue during the policy execution interval only. Nevertheless, the suitability of such implementation to the proposed algorithms remains to be evaluated.

Moreover, freshness-driven caching for dynamic content [30] is gaining increasing attention as several content types receive dynamic updates, e.g., news, weather, social media updates, etc., that render cached data irrelevant and prompt sophisticated caching decisions. This way, policies designed to handle dynamic content updates are also prone to require particular approaches since they may account for the content generation dynamics. For instance, the simple alternance with other policies would maintain stale versions of the same content in the cache.

Regarding the meta-policy strategy suitability for caches operating in different network systems, we believe the core principle of online learning the best policy through the perspective of a meta-policy is applicable for other cache systems besides ICNs, such as Content Delivery Networks

(CDNs), and Web proxy caches. Nevertheless, different systems may require adaptations to accommodate their particularities. For example, in our model, we represented the cache operating with total capacity because we have considered the plain on-path caching approach on the vast amount of content circulating the Internet. This way, the replacement policy is constantly in operation. That may not be a reality for CDNs working with prefetching strategies. Filling the cache with content is not always the optimal strategy because it has to consider the cost of moving content from the back-end database to the edge cache. The viability in such a scenario requires further investigation of cache management strategies in CDNs.

VII. CONCLUSION AND FUTURE WORK

In-network caching architectures can employ different cache replacement policies for each networking node. Configuring caching networks with appropriate policies is of fundamental importance to effectively realize the benefits of caching content. We have introduced a meta-policy approach that models the replacement policy choosing problem as a bandit problem. Such online learning techniques open up straightforward mappings to build self-driven intelligent networks. Our experimental results with traditional caching policies and basic non-stationary stochastic UCB algorithms revealed the potential for widespread use in different caching scenarios.

In future work, we intend to extend the model for collaborative cache systems in multi-cache networks. Collaborative caching systems are complex and may employ different caching strategies, but are crucial to obtain optimal/suboptimal overall network performance. One approach to tackle the correlation in cache's decisions is to model the problem as a combinatorial MAB (CMAB). In CMAB, a bandit plays a set of arms together and observes their individual rewards. This way, one action corresponds to a combination of different arms. The learning process, thus, aims to converge to the best combination. Such a strategy has been shown effective for proactive cache content placement in mobile BSs [14]. In that case, the contents are arms for a BS bandit player. The general problem is to choose the best combinations of contents to be cached over a fixed content set.

Regarding the caching replacement policy choosing problem for multiple caches, a possible combinatorial model would have a centralized entity deciding the policies' combinations for all caches. Instead of accounting for the individual cache efficiencies separately, the centralized entity would account for the aggregated network efficiency. A realistic assumption of collaborative caches would consider different caching settings, such as variations in cache sizes. An interesting aspect to investigate would be the impact of different cache sizes on the learning curve of the agents. Taking into consideration the replacement policies only and the on-path in-network caching approach explored in the experiments, a high variance among cache sizes would have an impact on the convergence time because the larger cache would naturally take longer to start executing content evictions and, therefore, support the learning process. Overall, that combinatorial model requires the study of computationally efficient multi-task bandit approaches.

Meanwhile, a centralized bandit convergence would be impractical in feasible times for dynamic, distributed, and heterogeneous caching settings with intermittent connections. Besides the variance of caching nodes, heterogeneous devices could work with different policy sets and bandit algorithms. Moreover, the caching nodes may employ different bandit iteration times. Such cache networks are well suited for collaborative multi-agent MAB models. Convergent learning is still challenging but feasible in combination with solutions designed for game-theoretical problems.

REFERENCES

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. 5th Int. Conf. Emerg. Netw. Exp. Technol.*, 2009, pp. 1–12.

[2] G. Gür, P. Porambage, and M. Liyanage, "Convergence of ICN and MEC for 5G: Opportunities and challenges," *IEEE Commun. Standards Mag.*, vol. 4, no. 4, pp. 64–71, Dec. 2020.

[3] R. Ullah, M. A. U. Rehman, M. A. Naeem, B.-S. Kim, and S. Mastorakis, "ICN with edge for 5G: Exploiting in-network caching in ICN-based edge computing for 5G networks," *Future Gener. Comput. Syst.*, vol. 111, pp. 159–174, Oct. 2020.

[4] S. Eum, Y. Shoji, M. Murata, and N. Nishinaga, "Design and implementation of ICN-enabled IEEE 802.11 access points as nano data centers," *J. Netw. Comput. Appl.*, vol. 50, pp. 159–167, Apr. 2015.

[5] M. Dräxler and H. Karl, "Efficiency of on-path and off-path caching strategies in information centric networks," in *Proc. IEEE Int. Conf. Green Comput. Commun.*, Nov. 2012, pp. 581–587.

[6] M. Gallo, B. Kauffmann, L. Muscariello, A. Simonian, and C. Tanguy, "Performance evaluation of the random replacement policy for networks of caches," in *Proc. 12th ACM SIGMETRICS/PERFORMANCE Joint Int. Conf. Meas. Modeling Comput. Syst.*, New York, NY, USA, 2012, pp. 395–396. [Online]. Available: http://doi-acm-org.ez10.periodicos.capes.gov.br/10.1145/2254756.2254810

[7] E. Newberry and B. Zhang, "On the power of in-network caching in the Hadoop distributed file system," in *Proc. 6th ACM Conf. Inf.-Centric Netw.*, 2019, pp. 89–99.

[8] S. Pires, A. Ziviani, and L. N. Sampaio, "Contextual dimensions for cache replacement schemes in information-centric networks: A systematic review," *PeerJ Comput. Sci.*, vol. 7, p. e418, Mar. 2021.

[9] P. Singh, R. Kumar, S. Kannaujia, and N. Sarma, "Adaptive replacement cache policy in named data networking," in *Proc. Int. Conf. Intell. Technol. (CONIT)*, Jun. 2021, pp. 1–5.

[10] S. Pires, A. Ribeiro, and L. Sampaio, "A meta-policy approach for learning suitable caching replacement policies in information-centric networks," in *Proc. 37th ACM/SIGAPP Symp. Appl. Comput.*, Apr. 2022, pp. 1950–1959.

[11] L. Zhang et al., "Named data networking (NDN) project," NDN Project Team, Univ. California, Los Angeles, CA, USA, Tech. Rep. NDN-0001, Oct. 2010. [Online]. Available: https://named-data.net/techreport/TR001ndn-proj.pdf

[12] I. U. Din, S. Hassan, M. K. Khan, M. Guizani, O. Ghazali, and A. Habbal, "Caching in information-centric networking: Strategies, challenges, and future research directions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1443–1474, 2nd Quart., 2017.

[13] O. Hahm, E. Baccelli, T. Schmidt, M. Wahlisch, and C. Adjih, "A named data network approach to energy efficiency in IoT," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2016, pp. 1–6.

[14] P. Blasco and D. Gündüz, "Learning-based optimization of cache content in a small cell base station," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 1897–1903.

[15] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Smart caching in wireless small cell networks via contextual multi-armed bandits," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.

[16] C. Zhang, P. Ren, and Q. Du, "A contextual multi-armed bandit approach to caching in wireless small cell network," in *Proc. 9th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Oct. 2017, pp. 1–6.

[17] T. Chen, Y. Du, B. Dong, Y. Chen, and C. Zhang, "Multi-objective learning approach to proactive caching in wireless networks," *IEEE Commun. Lett.*, vol. 23, no. 9, pp. 1538–1541, Sep. 2019.

[18] S. A. Bitaghsir, A. Dadlani, M. Borhani, and A. Khonsari, "Multi-armed bandit learning for cache content placement in vehicular social networks," *IEEE Commun. Lett.*, vol. 23, no. 12, pp. 2321–2324, Dec. 2019.

[19] C. Dai, K. Zhu, R. Wang, and B. Chen, "Contextual multi-armed bandit for cache-aware decoupled multiple association in UDNs: A deep learning approach," *IEEE Trans. Cognit. Commun. Netw.*, vol. 5, no. 4, pp. 1046–1059, Dec. 2019.

[20] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to cache with no regrets," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 235–243.

[21] J. Li, S. Shakkottai, J. C. S. Lui, and V. Subramanian, "Accurate learning or fast mixing? Dynamic adaptability of caching algorithms," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1314–1330, Jun. 2018.

[22] S. Mastorakis, A. Afanasyev, and L. Zhang, "On the evolution of ndnSIM: An open-source simulator for NDN experimentation," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 3, pp. 19–33, Jul. 2017.

[23] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "NDNSIM 2: An updated NDN simulator for NS-3," NDN, Univ. California, Los Angeles, CA, USA, Tech. Rep. NDN-0028, Revision 2, Nov. 2016.

[24] P. Gawłowicz and A. Zubow, "NS-3 meets OpenAI gym: The playground for machine learning in networking research," in *Proc. 22nd Int. ACM Conf. Modeling, Anal. Simulation Wireless Mobile Syst.*, Nov. 2019, pp. 113–120.

[25] A. Garivier and E. Moulines, "On upper-confidence bound policies for switching bandit problems," in *Proc. Int. Conf. Algorithmic Learn. Theory*. Berlin, Germany: Springer, 2011, pp. 174–188.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018, pp. 30–36.

[27] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Watch global, cache local: YouTube network traffic at a campus network: Measurements and implications," *Proc. SPIE*, vol. 6818, Jan. 2008, Art. no. 681805.

[28] D. E. Cooperation. (1996). *Digital's Web Proxy Traces*. [Online]. Available: hftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html

[29] C. R. Cunha, A. Bestavros, and M. E. Crovella, "Characteristics of www client-based traces," Comput. Sci. Dept., Boston Univ., Boston, MA, USA, Tech. Rep. BUCS-1995-010, 1995.

[30] B. Abolhassani, J. Tadrous, A. Eryilmaz, and E. Yeh, "Fresh caching for dynamic content," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2021, pp. 1–10.

STÉFANI PIRES received the B.Sc. degree in computer science from the Federal University of Paraéba (UFPB), the M.Sc. degree in computer science from the Federal University of Campina Grande (UFCG), and the Ph.D. degree in computer science from the Federal University of Bahia (UFBA), Brazil, in 2022. She is an Associate Professor of Technology at the Federal Institute of Bahia (IFBA). In 2007, she was a Sun Campus Ambassador with Sun Microsystems, providing training on open-source technologies. She is an SBC (the Brazilian Computer Society) member and holds one patent. Her current research interests include applied computing, machine learning, and content-centric networks.

ADRIANA RIBEIRO is a Ph.D. student in computer science with the Federal University of Bahia (UFBA). She is a Network Analyst at the Point of Presence of National Research Network in Bahia (PoP-BA/RNP). She integrates the Infrastructure and Systems for Networks and Telecom (INSERT) group, where she conducts research involving the areas of computer networks and future internet. In addition, she is a member of Digital Girls (Bahia Regional) and seeks to encourage the participation of women in the computer networks area.

LEOBINO N. SAMPAIO received the bachelor's and master's degrees in computer science from Salvador University (UNIFACS) and the Ph.D. degree from the Federal University of Pernambuco (UFPE). He is an Associate Professor with the Computer Science Department, Federal University of Bahia (UFBA). In 2020, he was a Visiting Researcher at UCLA. He received the CNPq Research Productivity Distinction Grant (Level 2) in 2022. He specializes in computer networks, focusing on network measurements and performance evaluation. Currently, he heads the INSERT group, is a Researcher for the Air Force Office of Scientific Research (AFOSR), and serves on several technical committees, including RNP's CT-Mon and the FIBRE Steering Committee. His research interests include information-centric networking (ICN), particularly the NDN architecture, and future internet architectures, such as SDN. He reviews for journals like IEEE *Communication Magazine* and *Computer Networks*.