# Online Caching Policy with User Preferences and Time-Dependent Requests: A Reinforcement Learning Approach

Mohammad Hatami[1], Markus Leinonen[1], Marian Codreanu[2]
[1]Centre for Wireless Communications, University of Oulu, Finland
[2]Department of Science and Technology, Linköping University, Sweden
Email: mohammad.hatami@oulu.fi, markus.leinonen@oulu.fi, marian.codreanu@liu.se

***Abstract*—Content caching is a promising approach to reduce data traffic in the back-haul links. We consider a system where multiple users request items from a cache-enabled base station that is connected to a cloud. The users request items according to the user preferences in a *time-dependent* fashion, i.e., a user is likely to request the next chunk (item) of the file requested at a previous time slot. Whenever the requested item is not in the cache, the base station downloads it from the cloud and forwards it to the user. In the meanwhile, the base station *decides* whether to replace one item in the cache by the fetched item, or to discard it. We model the problem as a Markov decision process (MDP) and propose a novel state space that takes advantage of the dynamics of the users' requests. We use reinforcement learning and propose a Q-learning algorithm to find an optimal cache replacement policy that maximizes the cache hit ratio without knowing the popularity profile distribution, probability distribution of items, and user preference model. Simulation results show that the proposed algorithm improves the cache hit ratio compared to other baseline policies.**

## I. Introduction

Wireless networks have been experiencing a huge increase in data traffic in recent years. However, despite the recent advances in mobile radio networks, these networks cannot keep up with the massive growth of data traffic [1]. Content caching has been envisioned to improve the efficiency in wireless content delivery by placing popular files close to the users to reduce data traffic in the back-haul links. Content caching can be divided into two main classes: 1) reactive *online* cache refreshment [2], where the base station refreshes the cache contents *on the fly* using the items fetched from the cloud during the interactions between the base station and the users, and 2) proactive *offline* cache refreshment [3]–[7], where the cache is updated only during dedicated time intervals (e.g., off-peak periods). This paper deals with online caching.

We address an online cache refreshment problem in a system where multiple users request items from a cache-enabled base station that is connected to a cloud server via a (costly) back-haul link. We focus on two fundamental aspects in caching, which have been usually overlooked so far.

- *Online* cache replacement: whenever the requested item does not exist in the cache and the item needs to be fetched from the cloud, the cache controller must decide whether to (i) use the fetched item to refresh the cache by replacing one of the existing items, or (ii) discard the item. This online cache refreshment is obtained nearly at zero cost without allocating any additional resources, e.g., time interval or bandwidth.
- User request model: unlike the most existing works where the requests are generated only based on a popularity distribution, we consider a more realistic model where the users request items according to user preferences in a *time-dependent* fashion, i.e., a user is likely to request the next chunk of the file (e.g., a video or an audio file) requested at a previous time slot.

We model the problem as an MDP and propose a novel state space that takes advantage of the dynamics of the users' requests. We use reinforcement learning and propose a Q-learning algorithm to find an optimal caching policy that maximizes the cache hit ratio. Simulation results show that the proposed algorithm improves the cache hit ratio compared to other baseline policies.

**Related Works:** In [3], the authors provided a probabilistic model to synthesize user preferences from content popularity. They also investigated the gain of optimizing the caching policy by learning user preferences over content popularity. The authors in [4] optimized the caching strategies of the users and small base stations to minimize a system cost in a device-to-device heterogeneous network. Differently from [3] and [4], we assume that the content popularity is unknown.

Machine learning allows modern wireless caching networks to be predictive and proactive. In [8], the authors reviewed major families of machine learning algorithms with potential applications in edge caching. Recently, reinforcement learning has also been applied in caching problems [2], [5]–[7]. The works [5], [6], and [7] consider proactive caching by refreshing the cache during dedicated time intervals (e.g., off-peak periods). Reinforcement learning techniques for online cache refreshment have also been discussed in [2]. However, the above works did not consider either the time-dependency of user requests or the user preferences.

## II. System Model

### A. Network Model

We consider the system model illustrated in Fig. 1. We assume that a cache-enabled base station serves a set of

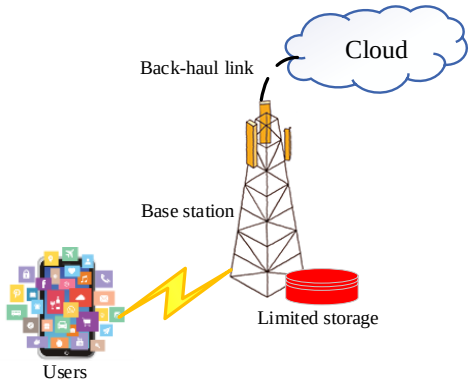


Fig. 1: System model

$\mathcal{K} = \{1, 2, \ldots, K\}$ users. The base station is connected to the cloud server via a back-haul link. The cloud server has a library of files $\mathcal{F} = \{1, 2, \ldots, F\}$, where each large file $f \in \mathcal{F}$ is divided into $L_f$ small chunks (items[1]). We assume that all the chunks in the library are of the equal size. The base station is equipped with a cache storage of limited size $M$, $M < \sum_{f=1}^{F} L_f$, i.e., the base station can store only $M$ items in its cache storage.

The system operates in a slotted fashion, i.e., time is divided into slots which are labeled with a discrete index $t \in \mathbb{N}$. We assume that at each slot, only one user requests an item and that request is served by the base station within the same slot. At each slot, a centralized controller selects a user randomly according to a user activity distribution $p_k$, $k \in \mathcal{K}$, $\sum_{k=1}^{K} p_k = 1$, where $p_k$ is the probability that user $k$ is selected to send its request at that slot.

Each user's request must be served either by fetching the requested item from the cloud or by utilizing the content currently available in the cache of the base station. If the requested item is not in the cache, the base station downloads the item from the cloud and serves the user's request. In this case, the base station will also decide if the downloaded item should be stored in the cache for its possible reuse in the future, i.e., the base station either replaces one item in the cache storage by the downloaded item, or discards it.

Let $a_t \in \mathcal{A}$ denote the caching action of the base station at time $t$, where $\mathcal{A} = \{0, 1, \ldots, M\}$ is the action space; 0 means that the fetched item is discarded whereas each $m$, $m = 1, \ldots, M$, means that the fetched item replaces the $m$th item in the cache storage. At each time slot $t$, the caching action $a_t \in \mathcal{A}$ is selected according to a policy. Accordingly, we aim to find the best policy to maximize the long-term cache hit ratio. The cache hit ratio is the ratio of the number of requests directly answered by the base station using the cache storage to the total number of requests. Thus, it measures how effective the caching policy is in reducing the data traffic in the back-haul link.

[1]In this paper, we use the words "item" and "chunk" interchangeably.

### B. User Request Model

In practice, if a user requests one chunk of a file (e.g., a video or an audio file) at time slot $t$, the same user is likely to request the next chunk of that file at a subsequent time slot. Formally, let $\zeta$ be the *continuation probability*, i.e., the probability that a user continues selecting the next chunk of the file that was chosen in its previous request. Suppose that user $k$ is selected to send its request and $l$th item of file $f$ was selected in its previous request. Then,

- If $l < L_f$, there are two cases: 1) with probability $\zeta$, user $k$ selects $(l+1)$th chunk of file $f$, and 2) with probability $1 - \zeta$, user $k$ changes the file, i.e., it first chooses a file index $f$ according to $p_{f|k}$ and then a chunk index $l$ according to $p_{l|k,f}$. The conditional distribution $p_{f|k}$, $f \in \mathcal{F}$, is called user preference. We assume that both $p_{f|k}$ and $p_{l|k,f}$ are not known to the base station.
- If $l = L_f$, user $k$ first chooses a file based on $p_{f|k}$ and then selects $l$th chunk of the file with probability $p_{l|k,f}$.

## III. Reinforcement Learning Based Online Caching Policy

### A. MDP Modeling and Problem Formulation

We model the online cache replacement as a Markov decision process (MDP) and, consequently, search for the best policy to maximize the cache hit ratio using reinforcement learning [9]. The MDP model is specified by the tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{P}(s_{t+1}|s_t, a_t), R_t, \gamma\}$, where

- $\mathcal{S}$ is the set of system states. The state at time slot $t$, denoted as $s_t$ contains the *request history* over past $H$ slots of 1) the $M$ items currently stored in the cache, and 2) the currently requested item. More precisely, $s_t = (\mathbf{h}_{t,0}, \mathbf{h}_{t,1}, \ldots, \mathbf{h}_{t,M})$, where $\mathbf{h}_{t,m} = [h_{t,m,1}, \ldots, h_{t,m,H}] \in \mathbb{B}^H$ is a binary vector. If $h_{t,m,i} = 1$, $m = 1, \ldots, M$, $i = 1, \ldots, H$, the current cache content $m$ was requested at slot $t - i$; otherwise $h_{t,m,i} = 0$. Similarly, $h_{t,0,i} = 1$ means that the currently requested item was (also) requested at slot $t - i$. It is important to point out that in order to keep track of the system state we need an additional table that stores the request history of *all* $\sum_{f=1}^{F} L_f$ items during the previous $H$ slots. By representing this table as a matrix of size $\sum_{f=1}^{F} L_f \times H$, a state $s_t$ consists of $M + 1$ rows of this matrix. Note that the state space is designed to take advantage of the dynamics of user requests. Namely, the state at each slot not only includes the rate of requests for the items but also incorporates the *exact* time slots when the items were requested during the past $H$ slots. Thus, this information represents the dynamic features of time-dependent requests along time.
- $\mathcal{A} = \{0, ..., M\}$ is the set of caching actions described in Section II-A. The action selected by the base station at time slot $t$ is denoted by $a_t$.
- $\mathcal{P}(s_{t+1}|s_t, a_t)$ is the state transition probability that maps a state-action pair at time step $t$ onto a distribution of states at time step $t + 1$.

- $R_t$ is the immediate reward function, where $R_t = 1$ when a cache hit occurred, otherwise $R_t = 0$.
- $\gamma \in (0, 1]$ is the discount factor. It is used to weight the immediate reward relative to future rewards. Typically, we set $\gamma < 1$ to guarantee that the cumulative reward is finite given that the immediate reward is bounded [9].

The long-term accumulated reward is defined as $G_t = \sum_{\tau=0}^{\infty} \gamma^\tau R_{\tau+t}$. We aim to maximize the expected long-term accumulated reward, $\mathbb{E}[G_t] = \mathbb{E}[\sum_{\tau=0}^{\infty} \gamma^\tau R_{\tau+t}]$, which is equal to maximizing the expected long-term cache hit ratio. The policy $\pi = \pi(a_t|s_t)$ is defined as a mapping from state $s_t$ to a probability of choosing action $a_t$. Thus, our goal is to find an optimal policy,

$$\pi_* = \arg\max_{\pi} \mathbb{E}_\pi \left[ \sum_{\tau=0}^{\infty} \gamma^\tau R_{\tau+t} \mid \pi \right] \tag{1}$$

### B. State-value and Action-value Functions

State-value and action-value functions are defined to evaluate the policy $\pi$. The state-value function of a state $s$ under a policy $\pi$, denoted by $v_\pi(s)$, is the expected return when starting in state $s$ and following the policy $\pi$ thereafter,

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[ \sum_{\tau=0}^{\infty} \gamma^\tau R_{\tau+t} | s_t = s \right], \forall s \in \mathcal{S}. \tag{2}$$

The action-value function, denoted by $q_\pi(s, a)$, represents the value of taking action $a$ in state $s$ under the policy $\pi$ and calculated as the expected return starting from $s$, taking the action $a$, and thereafter following the policy $\pi$,

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[ \sum_{\tau=0}^{\infty} \gamma^\tau R_{\tau+t} | s_t = s, a_t = a \right], \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{3}$$

The optimal action-value function for state $s$ and action $a$ is defined as $q_*(s, a) \doteq \max_\pi q_\pi(s, a)$. If $q_*(s, a)$ is available, the optimal policy $\pi_*$ is obtained simply by choosing action $a$ that maximizes $q_*(s, a)$ in each state.

The state-value and action-value functions in (2) and (3) can be rewritten in a recursive fashion as

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a)(r + \gamma v_\pi(s')), \tag{4}$$

$$q_\pi(s, a) = \sum_{s'} p(s'|s, a) \left( r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right), \tag{5}$$

where $p(s'|s, a) \in \mathcal{P}(s_{t+1}|s_t, a_t)$ is the probability of transferring to state $s'$ when choosing action $a$ in state $s$. The set of linear equations in (4) and (5) are known as the Bellman equations for state-value and action-value functions, respectively. The Bellman equations express a relationship between the value of a state and the values of its successor states.

The distribution of probabilities $p(s'|s, a)$ is determined by state transition probabilities $\mathcal{P}(s_{t+1}|s_t, a_t)$. If $\mathcal{P}(s_{t+1}|s_t, a_t)$ is available, we can use dynamic programming to find the optimal policy by using value iteration and policy improvement [9, Chapter 4]. These kinds of methods operating based on models are called model-based methods. When the model of $\mathcal{P}(s_{t+1}|s_t, a_t)$ is not available, we must use model-free reinforcement learning and learn the state-value and action-value functions by experience.

**Algorithm 1** Online Caching Policy via $\epsilon$-greedy Q-learning

**Initialize** $s_0$ randomly and $Q(s, a) = 0, \forall s, a$
**for** $t = 1, 2, 3, \ldots$ **do**
  **if** the requested item exists in the cache **then**
    $a_t = 0$, $R_t = 1$
  **else**
    $a_t$ is chosen according to the following probability

$$a_t = \begin{cases} \arg\max_{a \in \mathcal{A}} Q(s_t, a) & \text{, w.p. } 1 - \epsilon_t \\ \text{a random action } a \in \mathcal{A} & \text{, w.p. } \epsilon_t \end{cases}$$

    $R_t = 0$
  **end if**
  wait for the next request and compute $s_{t+1}$
  update
  $Q(s_t, a_t) \leftarrow (1 - \alpha_t) Q(s_t, a_t) + \alpha_t (R_t + \gamma \max_a Q(s_{t+1}, a))$
**end for**

### C. Q-learning Based Online Caching Policy

Q-learning is an online model-free reinforcement learning algorithm that directly estimates the $q_*(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}$, and finds the optimal policy iteratively. In the Q-learning method, the learned action-value function, $Q$, directly approximates the optimal action-value function $q_*$. However, all that is required for correct convergence, i.e., $Q \rightarrow q_*$, is that all the state-action pairs continue to be updated. To satisfy this condition, typically it is proposed to use the "exploration-exploitation" method for action selection. The $\epsilon$-greedy algorithm is one of the algorithms for trading off exploration and exploitation [9, Sect. 6.5].

Our proposed Q-learning based method is shown in Algorithm 1. The base station updates the estimated $Q(s_t, a_t)$ in each iteration. In each step that the requested item exists in the cache, the optimal action is $a_t = 0$, i.e., we do not have any changes in the cache contents. Otherwise, if the requested item does not exist in the cache, one can either take a random action or choose the greedy action. This algorithm more or less mimics a greedy algorithm as it generally exploits the best available option, but every once in a while the $\epsilon$-greedy algorithm explores the other available options. In Algorithm 1, the probability of taking a random action at time slot $t$ is denoted as $\epsilon_t$, and thus the probability of exploiting the greedy action is $1 - \epsilon_t$. In other words, at time slot $t$, the exploitation happens with probability $1 - \epsilon_t$, i.e., $a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a)$, and the exploration occurs with probability $\epsilon_t$, where the base station takes a random action $a \in \mathcal{A}$. Parameter $\epsilon_t$ determines the trade-off between exploration and exploitation. During initial iterations, it is in general beneficial to allow more exploration, i.e., to set $\epsilon_t$ high, in order to learn the underlying dynamics. On the other hand, in stationary settings and once enough observations are made, small values of $\epsilon_t$ are more preferable, i.e., increase tendency to exploitation.

## IV. Simulation Results

In this section, simulation results are presented to demonstrate the benefits of the proposed Q-learning caching policy.

### A. Simulation Setup

The simulation scenario consists of a base station and $K = 3$ users. The library contains $F = 10$ files. The popularity distribution for files is modeled by the Zipf distribution with parameter $\beta$. Thus, the probability that file $f$ is requested is $p_f = \frac{f^{-\beta}}{\sum_{f'=1}^{F} f'^{-\beta}}, 1 \leq f \leq F$, where the files are indexed in the descending order of popularity. When $\beta = 0$, the Zipf distribution is the same as the uniform distribution.

For simplicity, we assume that each file is divided into $L_f = 3$ chunks. In practice, when a user chooses a file according to its preference, the user is more likely to start from the first chunk of that file. On this account, we model the probability distribution of items by the Zipf distribution with parameter $\beta_{f,k}$ as $p_{l|f,k} = l^{-\beta_{f,k}} / \sum_{l'=1}^{L_f} l'^{-\beta_{f,k}}, 1 \leq l \leq L_f$. Note that other popularity profile distributions, kernel functions, and probability distribution of items are also applicable since the Q-learning approach learns the features from the history of requests to find the optimal policy.

To link the user preferences to the popularity distribution of the files, we model the joint probability that file $f$ is chosen by user $k$ as in [3], i.e., $p_{k,f} = p_f \frac{g(X_k, Y_f)}{\sum_{k'=1}^{K} g(X_{k'}, Y_f)}$, where $p_f$ is the popularity distribution of the files, $X_k$ and $Y_f$ are chosen uniformly random from $[0, 1]$, and $g(X_k, Y_f) \in [0, 1]$ is a kernel function used to control the correlation between the $k$th user and the $f$th file. The higher the value of $g(X_k, Y_f)$ the higher the probability that user $k$ requests file $f$. When $g(X_k, Y_f) = 0$, the $f$th file will never be chosen by the $k$th user. Various kernel functions such as power and Gaussian kernels can be applied. We use the power kernel function that is given as $g(X_k, Y_f) = (1 - |X_k - Y_f|)^{\left(\frac{1}{\mu^3} - 1\right)} \in [0, 1]$, $0 < \mu < 1$, where $\mu$ is a parameter that controls the average similarity among the user preferences. We set $\mu = 0.3$. Note that the user activities are obtained as the marginal distribution, i.e., $p_k = \sum_{f \in \mathcal{F}} p_{k,f}$, and user preferences are obtained as the conditional probabilities, i.e., $p_{f|k} = \frac{p_{k,f}}{p_k}$, $f \in \mathcal{F}$, $k \in \mathcal{K}$.

We evaluate the performance of the proposed algorithm in terms of cache hit ratio. Four common baseline caching policies are considered: 1) least recently used (LRU), 2) least frequently used (LFU), 3) first-in-first-out (FIFO), and 4) random policy. In the LRU policy, the base station keeps track of the most recent requests for every cached item. The cached item that is least recently requested will be replaced by the new item. In the LFU policy, the base station keeps track of the number of total requests for every cached item. The cached item that is requested the least many times will be replaced by the new item. In the FIFO policy, the base station keeps track of the time at which the item was cached; when the cache storage is full, the cached item that was stored the earliest will be replaced by the new item.

### B. Results

Fig. 2 depicts the learning curves of each algorithm for $\beta = 0.5$, $\beta_{f,k} = 0.5$, $\zeta = 0.7$, and cache storage capacity $M = 3$. As we can see in Fig. 2, the proposed Q-learning algorithm outperforms the other methods. However, the proposed algorithm does not benefit initially. This behavior is due to the characteristic exploration-exploitation trade-off of the Q-learning method, as discussed in Section III-C. In particular, during the first $10^5$ iterations, we set $\epsilon_t = 0.95$ to allow more exploration and after that we set $\epsilon_t = 0.05$ to allow more exploitation. Additionally, Fig. 2 illustrates the effect of the length of request history $H$ on the system performance. As $H$ increases, the gap between the proposed method and other baseline policies increases. This is because the developed algorithm utilizes more information at each state, improving the cache hit ratio. Note that the enhanced performance comes at the cost of complexity: when $H$ increases, the cardinality of the state space increases exponentially, commonly known as the curse of dimensionality.

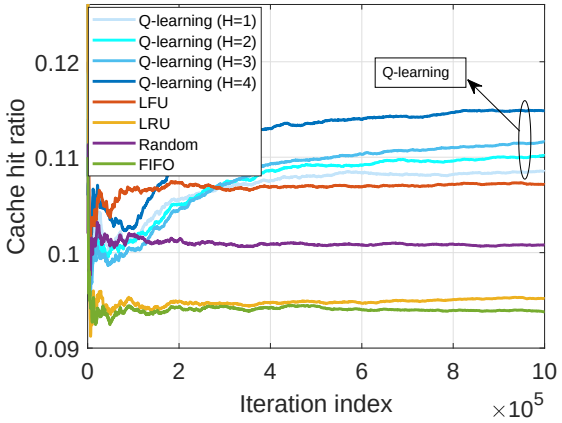


Fig. 2: Convergence behavior of the proposed algorithm and baseline methods in terms of cache hit ratio for $\beta = 0.5$, $\beta_{f,k} = 0.5$, $\zeta = 0.7$, and cache storage capacity $M = 3$.

Next, we focus on the average cache hit ratio obtained by averaging each algorithm over 10 episodes whereas each episode takes $5 \times 10^6$ iterations. For the rest of the results, we use $H = 5$ for our proposed algorithm. Fig. 3 illustrates the average cache hit ratio obtained by our Q-learning algorithm in comparison with the baseline methods for different values of parameter $\beta$. Our proposed algorithm outperforms the baseline methods for all values of $\beta$. Recall that $\beta = 0$ means uniform distribution for popularity. When $\beta$ increases, the average cache hit ratio increases as well, because the Zipf distribution becomes more concentrated so that the first few popular files account for the majority of requests. Thus, by storing the popular chunks of the popular files in the cache, more and more requests are answered by using the cache storage.

Fig. 4 shows the effect of the cache storage capacity $M$ on the long-term cache hit ratio. As expected, by increasing the cache storage capacity $M$, the cache hit ratio increases because more items can be stored in the cache.

Fig. 5 illustrates the impact of the continuation probability $\zeta$ on the average cache hit ratio. When $\zeta = 0$, each item becomes selected at a time slot $t$ merely based on the file and item popularity distributions that are concentrated towards the

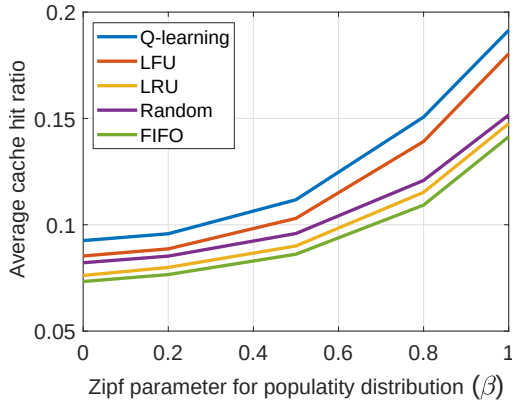


Fig. 3: Average cache hit ratio obtained by our Q-learning algorithm (with $H = 5$) in comparison with other baseline methods as a function of $\beta$ when $M = 3$, $\beta_{f,k} = 0.5$, and $\zeta = 0.7$.

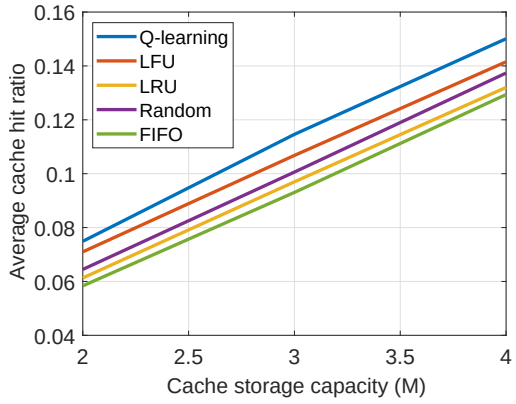


Fig. 4: Average cache hit ratio obtained by our Q-learning algorithm (with $H = 5$) in comparison with other baseline methods for different values of cache storage capacity $M$ for $\beta = 0.05$, $\beta_{f,k} = 0.5$, and $\zeta = 0.7$.

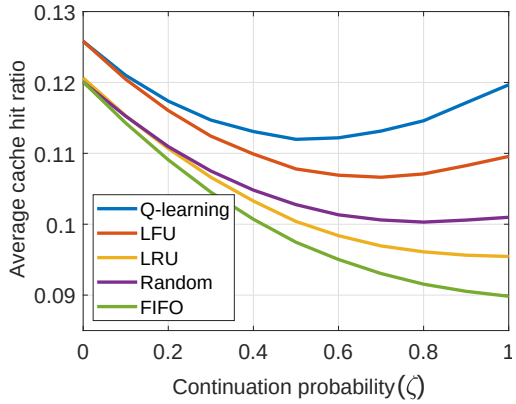


Fig. 5: Average cache hit ratio obtained by our Q-learning algorithm in comparison with other baseline methods as a function of $\zeta$ when $M = 3$, $\beta = 0.5$, and $\beta_{f,k} = 0.5$.

first files and chunks according to Zipf parameters $\beta = 0.5$ and $\beta_{f,k} = 0.5$, respectively. As $\zeta$ slightly increases, the users do not only tend to the most popular items, but occasionally, continue requesting the successive chunks of a file. Consequently, the predictability of the requests decreases and, thus, the average cache hit ratio of all methods decreases. On the other hand, when $\zeta$ keeps increasing and eventually reaches $\zeta = 1$, the users always continue selecting a next chunk of the previously chosen file, which again decreases the uncertainty of requests. For these high values of $\zeta$, since the developed algorithm can learn the request patterns with the aid of the proposed state definition, it performs significantly better than the baselines.

## V. Conclusions

In this paper, we modeled an online cache replacement problem as an MDP and proposed a Q-learning method based on the reinforcement learning framework to find the optimal caching policy to maximize the cache hit ratio. The proposed scheme does not need any information about the popularity profile distribution, probability distribution of items, and user preferences model. Simulation results showed that our Q-learning based approach significantly improved the cache hit ratio in comparison to the baseline methods. Possible future directions include the approaches for tackling the curse of dimensionality, e.g., via deep Q-learning.

## VI. Acknowledgment

This research has been financially supported by the Infotech Oulu, the Academy of Finland (grant 323698), and Academy of Finland 6Genesis Flagship (grant 318927). M. Codreanu would like to acknowledge the support of the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 793402 (COMPRESS NETS).

## References

[1] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5G systems," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 131–139, Feb. 2014.

[2] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues," *IEEE Network*, vol. 32, no. 6, pp. 50–57, Nov. 2018.

[3] B. Chen and C. Yang, "Caching policy for cache-enabled D2D communications by learning user preference," *IEEE Trans. Commun.*, vol. 66, no. 12, pp. 6586–6601, Dec. 2018.

[4] G. Quer, I. Pappalardo, B. D. Rao, and M. Zorzi, "Proactive caching strategies in heterogeneous networks with device-to-device communications," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 5270–5281, Aug. 2018.

[5] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5G using reinforcement learning of space-time popularities," *IEEE J. Select. Topics Signal Processing*, vol. 12, no. 1, pp. 180–190, Feb. 2018.

[6] ——, "Optimal dynamic proactive caching via reinforcement learning," in *Proc. IEEE Works. on Sign. Proc. Adv. in Wirel. Comms.*, Kalamata, Greece, Jun. 25-28 2018.

[7] W. Jiang, G. Feng, S. Qin, T. S. P. Yum, and G. Cao, "Multi-agent reinforcement learning for efficient content caching in mobile D2D networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 3, pp. 1610–1622, Mar. 2019.

[8] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, "Learn to cache: Machine learning for network edge caching in the big data era," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 28–35, Jun. 2018.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.