

Deep Reinforcement Learning Based Edge Caching in Wireless Networks

Chen Zhong, M. Cenk Gursoy, and Senem Velipasalar

Abstract—With the purpose to offload data traffic in wireless networks, content caching techniques have recently been studied intensively. Using these techniques and caching a portion of the popular files at the local content servers, the users can be served with less delay. Most of the content replacement policies are based on the content popularity, that depends on the users' preferences. In practice, such information varies over time. Therefore, an approach to determine the file popularity patterns must be incorporated into caching policies. In this context, we study content caching at the wireless network edge using a deep reinforcement learning framework with Wolpertinger architecture. In particular, we propose deep actor-critic reinforcement learning based policies for both centralized and decentralized content caching. For centralized edge caching, we aim at maximizing the cache hit rate. In decentralized edge caching, we consider both the cache hit rate and transmission delay as performance metrics. The proposed frameworks are assumed to neither have any prior information on the file popularities nor know the potential variations in such information. Via simulation results, the superiority of the proposed frameworks is verified by comparing them with other policies, including least frequently used (LFU), least recently used (LRU), and first-in-first-out (FIFO) policies.

I. INTRODUCTION

The rapid growth in the number of mobile devices and in rich media-enabled applications has led to a 17-fold increase in mobile data traffic from 2012 to 2017 [1]. Mobile video accounts for more than half of this data traffic, and is predicted to further grow by 9-fold, accounting for 79% of the total data traffic, by 2022. However, the increase in mobile network connection speed, which is predicted to be only about three-fold, will not be adequate to satisfy the users' demands on high-quality streaming services.

To better serve the users, content caching strategies have been studied recently. In particular, content caching is considered as a key approach to reduce the data traffic as it enables the content server nodes to store a part of the popular contents locally, so that when the cached contents are requested, the server can deliver the content directly to the users and reduce the delay and congestion in the network. Motivated by this, different caching strategies have been studied in the literature. For central content servers, such as the baseband unit in a cloud radio access network (C-RAN), centralized coded caching and delivery schemes were

presented in [2] and [3]. Regarding decentralized caching, the authors in [4] presented a decentralized optimization method for the design of caching strategies that aimed at minimizing the energy consumption of the network. Recently, proactive caching at the wireless network edge, such as at the base stations and user equipments, is proposed. This technique makes it possible to have popular contents to be placed closer to the end users and be directly transmitted, which can effectively reduce the time compared to routing in content delivery networks (CDNs), and apparently save a considerable amount of waiting time for users and offload a portion of the data traffic at the CDN. For instance, authors in [5] and [6] studied edge caching policies aimed at minimizing the transmission delay in cellular networks. In [7], a decentralized framework for proactive caching was proposed based on blockchains considering a game-theoretic point of view. In [8], caching and multicast problems were jointly solved using dynamic programming. In addition, studies on hierarchical caching have also been conducted. For instance, in [9], [10], [11], hybrid content caching schemes for joint content caching control at the baseband unit and radio remote heads were presented. And the authors in [12] proposed an edge hierarchical caching policy for caching at small base stations and user equipments. In addition, other caching strategies have been intensively studied recently considering different models and strategies. For instance, the study in [13] proposed an age-based threshold policy which caches all contents that have been requested by more than a certain threshold. Furthermore, popularity-based content caching policies named StreamCache and PopCaching were studied in [14] and [15], respectively. Recently, femtocell caching [16], coded caching [17] and D2D caching [18] have also been investigated.

We in this work concentrate on edge caching at small base stations, where the caching policy at the base station is driven by the content popularity. Hence, content popularity is the key to solve the caching problem. In previous works, content popularity is assumed to be either known to the content server as presented in [7], [19], or estimated before the caching actions as proposed in [20], [6]. The former assumption makes the framework less practical when the content popularity is time-varying, and the estimation of the content popularity or the arrival intensity of the users' requests will lead to large overhead. To avoid these drawbacks, machine learning methods have recently been introduced to

The authors are with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, 13244 (e-mail: czhong03@syr.edu, mcgursoy@syr.edu, svelipas@syr.edu).

determine efficient caching policies. For instance, the authors in [21] trained the optimization algorithms for caching through a deep neural network in advance. In other studies, different deep reinforcement learning (DRL) algorithms were used to find the caching policies that can better adapt to changing environments. In [22], authors implemented a Q-learning algorithm to find the optimal caching policy. In [23] and [24], the use of actor-critic deep reinforcement learning frameworks for caching policies were studied. And for the cooperative caching policy in decentralized caching networks, a multi-agent Q-learning solution was proposed in [25], and authors in [26] and [27] presented two different multi-armed bandit based caching schemes.

In this work, we propose a DRL based framework for edge content caching at the base stations. We initially consider a single-cell wireless scenario with one base station, and study centralized caching where the single base station is the only cache-enabled content server. Subsequently, we address a multi-cell wireless network, and consider a decentralized caching framework, where each base station is equipped with caching storage space.

As seen in previous studies, content popularity distribution has a key role in content caching problems. Though the DRL algorithms do not require the content popularity information, the agent needs to observe enough features of the environment to ensure the accuracy of its decisions, so we adopt the Wolpertinger architecture based actor-critic DRL framework [28] to deal with large discrete action spaces. And for the decentralized caching system, we propose a multi-agent framework [29], [30], [31] for cooperative caching.

The main contributions of this paper are summarized below:

- We present an actor-critic DRL framework, which adopts the Wolpertinger architecture, for content caching in the case of a single base station, and extent it to a multi-agent actor-critic framework used for decentralized cooperative caching at multiple base stations.
- We analyze the performance of the proposed framework for centralized caching in terms of the cache hit rate, and provide comparisons with other caching policies including least recently used (LRU), least frequently used (LFU), and first-in first-out (FIFO) policies. We demonstrate that the DRL agent is able to achieve improved short-term cache hit rate and improved and stable long-term cache hit rate.
- We analyze the performance of the proposed multi-agent framework in terms of the cache hit rate and also the transmission delay reduction, and again provide comparisons with the LRU, LFU, and FIFO caching strategies. We show that the proposed multi-agent framework can identify the popular contents effectively, and outperform the other schemes.

The remainder of the paper is organized as follows. First, in Sections II and III, we focus on the study of a centralized caching system. Specifically, in Section II, the single-cell

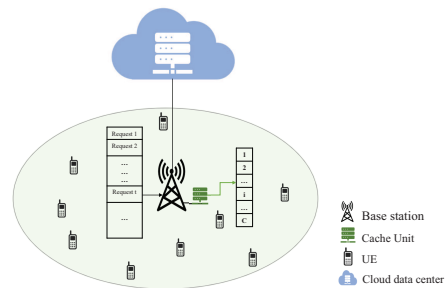


Fig. 1. System model of a centralized caching system.

system model and the cache hit rate maximization problem is introduced. In Section III, the deep actor-critic reinforcement learning based centralized edge caching framework is proposed. Subsequently, we extent our analysis to the decentralized edge caching scenario in Sections IV and V. In Section IV, we introduce the multi-cell system and the decentralized edge caching problems based on the overall cache hit rate and transmission delay. And in Section V, we demonstrate the multi-agent framework for the decentralized edge caching scenario. Numerical results are presented and discussed in Section VI. Finally, we conclude in Section VII.

II. CENTRALIZED EDGE CACHING IN A SINGLE-CELL NETWORK: SYSTEM MODEL AND PROBLEM FORMULATION

A. System Model

In the centralized caching system, shown in Fig. 1, we assume that there is only one cache-enabled base station, which is the content server for all users in its coverage. It is also assumed that the total number of contents that can be requested by the user is M and the base station can store C contents at most¹. These contents have different popularities, which can be quantified by the probability that the content will be requested by the users. In this centralized caching system, we assume that the requests from users arrive at the base station one by one, and a Zipf distribution is used to approximatively describe the popularity distribution of the files at all users. Here, we assign each content a unique index, and use this index as the content ID when users request the content. So, we can denote the requests from the users as $Req = \{R_1, R_2, R_3, \dots\}$, where R_t denotes the ID of the requested content at time t .

Since the base station is equipped with cache storage, it first checks if the contents requested by the users are cached locally. If the requested contents are available in the local cache, then the base station can transmit the contents to the corresponding user without requesting them from the upper level content servers. To avoid requesting content as much as possible, the base station needs to update its cache according

¹In this work, we assume that all contents have the same size. And based on this assumption, we use the number of contents, C , that can be stored at a base station as the cache capacity.

to users' preferences. Each time a request arrives, the base station, as noted above, will first check if the requested content is available locally, so that it can decide how to serve the user. Then, the base station has to decide whether or not to update its cache. Therefore, for each content, there are two cache states: cached, and not cached. The cache state gets updated based on the caching decision. Here, we define two types of actions: the first one is to find a pair of contents and exchange the cache states of the two contents and the second one is to keep the cache states of all the contents unchanged. We describe the action space at the base station as $\mathcal{A} = \{a_1, a_2, a_3, \dots, a_m\}$, where $a_v, v = 1, 2, \dots, m$, denotes a valid action and m is the size of the action space. Note that the replacement in the cache requires two contents in pair: one is to be added to the cache, and the other one is to be removed from the cache. Hence, the value of m depends on the cache capacity and the number of content files, and has a finite but generally a large value. Theoretically, multiple actions can be executed in one decision epoch. To reduce the computational complexity, we need to limit the action space size m and the number of actions to be executed in one decision epoch. For this purpose, we propose to employ the Wolpertinger architecture based DRL framework as will be discussed in detail in subsection III-B.

B. Problem Formulation

In this part, we formulate the caching problem with the objective to maximize the cache hit rate, which describes how frequently the requested content is found in the local cache.

For the centralized caching system, the cache hit rate is computed from the perspective of the base station, and the cache hit rate P_{hit}^c in T requests is defined as

$$P_{hit}^c = \frac{\sum_{t=1}^T \mathbf{1}(R_t)}{T} \quad (1)$$

where indicator function $\mathbf{1}(R_t)$ is defined as

$$\mathbf{1}(R_t) = \begin{cases} 1, & \text{if the request } R_t \text{ hits in the cache,} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

with the cache being essentially described as the set of indices of the contents in the cache at time t (with cardinality equal to the cache capacity C).

And the problem of the maximization of the cache hit rate over the caching states can be expressed as

$$\mathbf{P1:} \quad \underset{\Phi}{\text{Maximize}} \quad P_{hit}^c \quad (3)$$

$$\text{Subject to} \quad \sum_{f=1}^M \phi_f \leq C \quad (4)$$

where Φ is the $1 \times M$ dimensional content state vector that records the states of all contents (describing whether they

are cached or not), and each element ϕ_f in the content state vector is an indicator to show if the file is cached:

$$\phi_f = \begin{cases} 1 & \text{if the file } f \text{ is cached at the base station} \\ 0 & \text{if the file } f \text{ is not cached at the base station} \end{cases} \quad (5)$$

III. DEEP ACTOR-CRITIC FRAMEWORK FOR CENTRALIZED EDGE CACHING

To solve the optimization problem **P1**, we in this section propose a Wolpertinger architecture based single-agent actor-critic DRL framework for centralized edge caching. First, we introduce the related definitions in this architecture. Several of these definitions will also be used for the decentralized edge caching framework in Sections IV and V.

A. Related Definitions

1) Agent's Observation and State Space:

a) *Observations of the Centralized DRL Agent:* The centralized DRL caching agent assumes the feature space of the cached contents and the currently requested content as the state. In each decision epoch, we assign a temporary index to each content from which we need to extract features. Since we only extract the features from cached contents and the currently requested content, we let the indices range from 0 to cache capacity C . The index of the currently requested content is 0, while the index of the cached content varies from 1 to C . This temporary index is different from the content ID and is only used for denoting the feature. Thus, the observed state at time t is defined as $s_t = \{\mathcal{F}_s; \mathcal{F}_m; \mathcal{F}_l\}$, where $\mathcal{F}_s, \mathcal{F}_m, \mathcal{F}_l$ are the features collected at different times, as will be discussed next.

b) *Feature Space:* The feature space consists of three components: short-term feature \mathcal{F}_s , medium-term feature \mathcal{F}_m , and long-term feature \mathcal{F}_l , which represent the total number of requests for each content in a specific short-, medium-, long-term, respectively. These features are updated as new requests arrive at agents. Then, we let f_{xj} , for $x \in \{s, m, l\}$ and $j \in \{1, \dots, M\}$, denote the feature of a specific content within a specific term, where M is the total number of contents. As mentioned above, the observation for each agent can be expressed by $\{\mathcal{F}_s; \mathcal{F}_m; \mathcal{F}_l\}$, and we have $\mathcal{F}_s = \{f_{s0}, f_{s1}, \dots, f_{sM}\}$, $\mathcal{F}_m = \{f_{m0}, f_{m1}, \dots, f_{mM}\}$, and $\mathcal{F}_l = \{f_{l0}, f_{l1}, \dots, f_{lM}\}$.

2) *Action Space:* In order to limit the action size, we restrict that the DRL agent can either only replace one selected cached content by the currently requested content, or keep the cache state the same. Thus, each replacement action of the caching agents indicates a pair of content IDs: one is the ID of the cached content to be deleted from the cache, and the other one is the ID of the content which is currently requested. So, all possible replacement actions can be described by a $C \times L$ matrix, where C is the cache capacity and L is the number of requests arriving simultaneously. For the centralized caching agent, since we assume that the users'

requests arrive one by one, we have $L = 1$, which means that the replacement decision must be made between only the current requested content and a cached content. Therefore, the action space of the centralized caching agent can be defined as $\mathcal{A} = \{0, 1, 2, \dots, C\}$, where C is again the cache capacity at the base station.

And we assume that only one action can be selected in each decision epoch. Let a_t be the selected action in epoch t . Note that for each caching decision, there are $(C+1)$ possible actions. When $a_t = 0$, the currently requested content is not stored, and the current caching space is not updated. And when $a_t = v$ for $v \in \{1, 2, \dots, C\}$, the action is to store the currently requested content by replacing the v^{th} content in the cache space.

3) *Reward*: As stated in the previous section, the centralized caching agent aims at maximizing the cache hit rate to solve problem **P1**. The reward r_t for each decision epoch depends on the short and long-term cache hit rate. For example, we set the short-term reward, considering the number of requests for local content in the next epoch, i.e., the short-term reward $P_{hit,s}^c$ can be either 0 or 1. And let the total normalized number of requests for local content within the next 100 requests as the long-term reward $P_{hit,l}^c \in [0, 1]$. The total reward for each step is defined as the weighted sum of the short and long-term rewards

$$r_t = P_{hit,s}^c + w * P_{hit,l}^c \quad (6)$$

where w is the weight to balance the short and long-term rewards. For instance, if we lower the value of w , we give more priority to the short-term reward to maximize the cache hit rate at every step given the chosen action.

B. Wolpertinger Architecture

Based on the Wolpertinger Policy [28], our framework consists of three main components: actor network, K-nearest neighbors (KNN), and critic network. We train the centralized caching policy using the Deep Deterministic Policy Gradient (DDPG) [32]. This Wolpertinger architecture works in three steps. First, the actor network takes cache state and the current content request as its input, and provides a single proto actor \hat{a} at its output. Then, KNN receives the single actor \hat{a} as its input, and calculate the L_2 distance between every valid action and the proto actor in order to expand the proto actor to an action space, denoted by \mathcal{A}_K , with K elements and each element being a possible action $a_v \in \mathcal{A}$. And at the last step, the critic network takes the action space \mathcal{A}_K as its input, and refines the actor network on the basis of the Q value. The DDPG is applied to update both critic and actor networks.

Below we provide a more detailed description of the key components of the algorithm.

The actor: The actor network is designed to choose a proto-actor $\hat{a} \in \mathcal{A}$ from the valid actions. This selection is based on the decision policy of the actor network, and will be updated after each decision.

K-nearest neighbors (KNN): The generation of the proto-actor can help reduce the potentially high computational complexity due to the large size of the action space. However, reducing the high-dimensional action space to one actor will lead to poor decision making. To remedy this, the K-nearest neighbors mapping, g_K , is applied to expand the actor \hat{a} to a set of valid actions selected from the action space \mathcal{A} . The set of actions returned by g_K is denoted by \mathcal{A}_K :

$$\mathcal{A}_K = g_K(\hat{a}_t) \quad (7)$$

where

$$g_K = \arg \min_{a \in \mathcal{A}}^K |a - \hat{a}|^2. \quad (8)$$

With (8), we determine the K nearest neighbors of the proto-actor. Here, a is a valid action in the action space \mathcal{A} , and $|a - \hat{a}|^2$ is the L_2 distance of the features between the action a and the proto-actor \hat{a} . When the proto-actor is selected by the actor network, the agent will traverse the action space to find the K nearest feature distances, and the action set will be determined accordingly.

The critic: To avoid the actor with low Q -value being occasionally selected, a critic network is defined to refine the actor. The critic network will evaluate all actions in the expanded action space, and the action that provides the maximum Q value will be chosen as a_t , i.e.,

$$a_t = \arg \max_{a_j \in \mathcal{A}_K} Q(s_t, a_j). \quad (9)$$

C. Single-Agent Actor-Critic Framework

In this subsection, we present the details of the single-agent actor-critic framework for the centralized caching system, and introduce the update of the two networks.

The actor: The actor network is defined as a function parameterized by θ^μ , mapping the state \mathcal{S} from the state space to the action space \mathcal{A} . The mapping provides a proto-actor \hat{a} in \mathcal{A} for a given state under the current parameter. Here, we scale the proto-actor to make sure \hat{a} is a valid action i.e., $\hat{a} \in \mathcal{A}$:

$$\mu(s|\theta^\mu) : \mathcal{S} \rightarrow \mathcal{A} \quad (10)$$

$$\mu(s|\theta^\mu) = \hat{a}. \quad (11)$$

The critic: The critic is employed as a refining network, and the deterministic target policy is described below:

$$Q(s_t, a_j|\theta^Q) = \mathbb{E}_{r_t, s_{t+1} \sim E} [r(s_t, a_j) + \gamma Q(s_{t+1}, a_{t+1}|\theta^Q)] \quad (12)$$

where θ^Q stands for the parameters of the critic network, and $\gamma \in (0, 1]$ is the discount factor which weighs the future accumulative reward $Q(s_{t+1}, a_{t+1}|\theta^Q)$. Here, the critic takes both the current state s_t and the next state s_{t+1} as its input to calculate the Q value for each action in \mathcal{A}_K . Then, the action that provides the maximum Q value will be chosen as a_t , i.e.,

$$a_t = \arg \max_{a_j \in \mathcal{A}_K} Q(s_t, a_j|\theta^Q). \quad (13)$$

Update: To update the parameters of actor and critic, we replay a minibatch of samples randomly selected from the previous transition, with a minibatch size N_B . Therefore, the actor policy is updated using deep deterministic policy gradient, which is given as

$$\nabla_{\theta^\mu} J \approx \frac{1}{N_B} \sum_i \nabla_a Q(s, a | \mu^Q) |_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s | \theta^\mu) |_{s_i} \quad (14)$$

and the critic is updated by minimizing the loss:

$$L = \frac{1}{N_B} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2 \quad (15)$$

where $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'}) | \theta^{Q'})$.

Workflow: In this part, we introduce the workflow of the proposed framework. At the beginning of each epoch t , the agent observes the state s_t from the environment. Then, the proto-actor obtained by the actor network based on the current policy will be passed to KNN, and expanded action set will be evaluated by the critic network. Then, an ϵ -greedy policy is applied at selecting the action a_t . This policy can force the agent to explore more possible actions. After the chosen action is executed in the environment, the transition (s_i, a_i, r_i, s_{i+1}) will be stored to the memory \mathcal{M} at the end of this epoch. Next, a minibatch with size N_B will be randomly sampled from the memory \mathcal{M} and replayed to update the actor and critic networks. The complete process is presented in Algorithm 1 below.

IV. DECENTRALIZED EDGE CACHING IN MULTI-CELL NETWORKS: SYSTEM MODEL AND PROBLEM FORMULATION

A. System Model

The decentralized caching system considered in this section is depicted in Fig. 2. The system consists of a cloud data center and N cache-enabled base stations. Similar to the centralized content server, each base station in this decentralized content caching system also has a fixed cache capacity C and is able to serve the users from the cache when the requested contents are available locally. For the contents not cached locally, a request is generated by the base station to retrieve the content from the cloud data center. Here, we assume that the cloud data center has sufficient storage space to have all content files, and all base stations can connect with the cloud data center. As shown in the figure, each base station covers a fixed cellular region described by a circle with the corresponding base station at the center, and the radii of the cells are fixed and all users in the cell can access the corresponding base station. There are U users randomly distributed in the system, and they are located in at least one cellular region covered by a base station to ensure service. Each base station receives requests from all users in its cellular region simultaneously, and based on the requests, the base station will learn users' preferences for contents and make caching decisions.

Algorithm 1 Single-Agent Actor-Critic Algorithm for Content Caching

- 1: Randomly initialize critic network $Q(s, a | \theta^Q)$ and actor $\mu(s | \theta^\mu)$ with weights θ^Q and θ^μ .
 - 2: Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
 - 3: Initialize replay buffer \mathcal{M} with capacity of $N_{\mathcal{M}}$
 - 4: Initialize features space \mathcal{F}
 - 5: **for** $t = 1, T$ **do**
 - 6: The base station receive a request R_t
 - 7: **if** Requested content is already cached **then**
 - 8: Update cache hit rate and end epoch;
 - 9: **else**
 - 10: **if** Cache storage is not full **then**
 - 11: Cache the currently requested content
 - 12: Update cache state and cache hit rate
 - 13: End epoch;
 - 14: **end if**
 - 15: Receive observation state s_t
 - 16: Actor: Receive proto-action from actor network $\hat{a}_t = \mu(s_t | \theta^\mu)$.
 - 17: KNN: Retrieve k approximately closest actions $\mathcal{A}_K = g_K(\hat{a}_t)$
 - 18: Critic: Select action $a_t = \arg \max_{a_j \in \mathcal{A}_K} Q(s_t, a_j | \theta^Q)$ according to the current policy.
 - 19: Execute action a_t , and observe reward r_t and observe new state s_{t+1}
 - 20: Store transition (s_t, a_t, r_t, s_{t+1}) in \mathcal{M}
 - 21: Sample a random mini batch of N_B transitions (s_i, a_i, r_i, s_{i+1}) from \mathcal{M}
 - 22: Set target $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'}) | \theta^{Q'})$
 - 23: Update critic by minimizing the loss: $L = \frac{1}{N_B} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2$
 - 24: Update the actor policy using the sampled policy gradient: $\nabla_{\theta^\mu} J \approx \frac{1}{N_B} \sum_i \nabla_a Q(s, a | \mu^Q) |_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s | \theta^\mu) |_{s_i}$
 - 25: Update the target networks with $\tau \ll 1$: $\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$
 - 26: $\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$
 - 27: Update the cache state
 - 28: Update features space \mathcal{F}
 - 29: Update cache hit rate
 - 30: **end if**
 - 31: **end for**
-

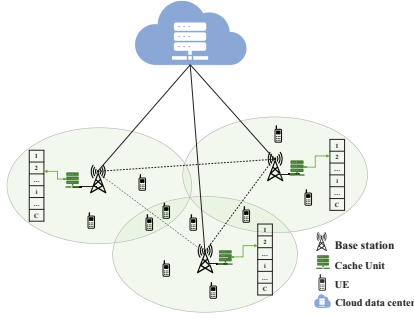


Fig. 2. System model of a decentralized caching system.

We assume that in a given time slot, the users' locations do not change and those located in the overlapped regions can be served by any one of the corresponding base stations. Users have their own preferences for contents, and in each time slot, each user can request only one content. Here, we denote the total number of contents as M , and use the content ID to denote the requests for the corresponding content. In each operation cycle, users request contents based on their own preferences. The requests are sent to all base stations that can connect with the user, and, for instance, when delay is the performance metric, the base station that provides the minimum transmission delay will finally transmit the requested content file to the user. In the meantime, all base stations will update their caches to improve the cache hit rate or minimize the average transmission delay based on the users' requests.

The base stations will compete with each other to get the chance to transmit and also cooperate with each other to reduce the overall transmission delay. To realize this framework, we propose a Wolpertinger architecture based multi-agent framework. In this framework, there are N actor networks and one centralized critic network. We consider each base station as an agent that adopts one of the actor networks to seek its own caching policy. And we assume there are control channels that allow the base stations to send the caching state and data traffic parameters to the cloud data center, so that the cloud data center can act as the centralized critic to evaluate the overall caching state². Similar to the centralized content server, in each operation cycle, the decentralized agent can either keep the cache state the same or replace unpopular contents with the popular ones. Note that there can be more than one request arriving at a base station at the same time, and for different contents, the agent needs to jointly decide which cached content will be deleted and which content requested by which user will be cached. For each agent i , we define the action space as \mathcal{A}_i , and let $\mathcal{A}_i = \{a_0, a_1, \dots, a_{\mathcal{D}_i}\}$, where a_ν denotes a valid action. In our case, a_0 indicates that the current cache state is

²We note that the centralized critic can also be placed at a node or controller (other than the cloud data center) that is connected to the BSs.

unchanged. For $\nu = \{1, 2, \dots, \mathcal{D}_i\}$, we define $\mathcal{D}_i = \binom{C_i}{1} \binom{L_i}{1}$ ⁶, where C_i is the total number of files that can be stored at base station i , and L_i is the number of users that can connect with the base station i . So each a_ν stands for a possible combination to replace one of C_i cached contents with one of L_i currently requested contents. In every time slot, each agent must select its own action from the corresponding action space \mathcal{A}_i and execute.

As seen in the descriptions above, decentralized caching in a multi-cell network is more general and challenging than the centralized caching with a single base station analyzed in the previous sections. In the decentralized caching framework, base stations receive multiple file requests at a given time and differentiate which users generate the requests, each user generates the file requests according to its unique preference, user location and channel conditions are taken into account if the objective is to reduce the transmission delay, and overall multi-agent reinforcement learning is employed. A challenge in this setting is that the action space and observation space grow as the number of users in the coverage of the base stations increases.

B. Problem Formulation

1) *Cache Hit Rate*: For the decentralized caching system, the cache hit rate is calculated from the perspective of users in each time slot t . In particular, the cache hit rate is defined as

$$P_{hit}^d = \frac{\sum_{j=1}^U \xi_j}{U} \quad (16)$$

where U is the total number of users, and ξ_j is an indicator defined as

$$\xi_j = \begin{cases} 1 & \text{if user } j \text{ is served by a base station} \\ & \text{(i.e., from the base station's local cache)} \\ 0 & \text{if user } j \text{ is served by the upper level server.} \end{cases}$$

And the maximization of the cache hit rate over the caching decisions is expressed as

$$\mathbf{P2:} \quad \underset{\Phi}{\text{Maximize}} \quad P_{hit} \quad (17)$$

$$\text{Subject to} \quad \sum_{f=1}^M \phi_{i,f} \leq C_i \quad \forall i \in \{1, \dots, N\} \quad (18)$$

where Φ is an $N \times M$ matrix which records the caching states of the N base stations, and each element $\phi_{i,f}$ in the caching state matrix is an indicator to show if the file is cached at base station i :

$$\phi_{i,f} = \begin{cases} 1 & \text{if the file } f \text{ is cached at the base station } i \\ 0 & \text{if the file } f \text{ is not cached at the base station } i \end{cases} \quad (19)$$

2) *Transmission Delay*: For the decentralized caching system, we evaluate the caching policy in terms of transmission delay as well. The transmission delay is defined as the number of time frames needed to transmit a content file, and can be expressed as

$$T = \min \left\{ \tilde{t} : F \leq \sum_{\kappa=1}^{\tilde{t}} T_0 \mathbb{C}[\kappa] \right\} \quad (20)$$

where F is the size of the content file to be transmitted. T_0 stands for the duration of each time frame, and $\mathbb{C}[\kappa]$ is the instantaneous channel capacity in the κ^{th} time frame. And the channel capacity $\mathbb{C}[\kappa]$ is expressed as

$$\mathbb{C}[\kappa] = B \log_2 \left(1 + \frac{P_t}{BN_0} z_\kappa \right) \quad \text{bits/s} \quad (21)$$

where P_t is the transmission power, B is the channel bandwidth, N_0 is the (one-sided) noise power spectral density, and z_κ is the channel gain in the κ^{th} time frame. In the system, there are two types of transmitters: the cloud data center and the base stations. We assume that all transmitters transmit at their maximum power level to maximize the transmission rate. The transmission power is denoted as

$$P_t = \begin{cases} P_c & \text{if the transmitter is the cloud data center} \\ P_i & \text{if the transmitter is the } i^{\text{th}} \text{ base station} \end{cases} \quad (22)$$

So, if user j requests a content, which is not cached at any base station that can connect with the user, the content file will be first transmitted from the cloud data center to the base station \hat{i} , which is the closest base station to the user j , and then from the base station \hat{i} to user j . Thus, the minimum transmission delay \hat{D}_j in the case of a missing file in the cache can be expressed as

$$\hat{D}_j = T_{c,\hat{i}} + T_{\hat{i},j} \quad (23)$$

where $T_{c,\hat{i}}$ stands for the transmission delay from the cloud data center to the base station \hat{i} , and $T_{\hat{i},j}$ is the transmission delay from the base station \hat{i} to the user j .

However, if the requested file is cached at a base station i , which can connect to user j , the transmission delay D_j in the case of having a hit in the cache can be expressed as

$$D_j = T_{i,j}. \quad (24)$$

Problem Formulation: In the previous section, we have described the transmission delay for both cases of missing and hitting in the cache. In this section, we formulate the caching problem. First, we define the transmission delay reduction ΔD_j as

$$\Delta D_j = \hat{D}_j - D_j. \quad (25)$$

Now, the average transmission delay reduction in an opera-

tion cycle is

$$\Delta D = \frac{1}{U} \sum_{j=1}^U \Delta D_j \quad (26)$$

$$= \frac{1}{U} \sum_{j=1}^U (\hat{D}_j - D_j) \quad (27)$$

$$= \frac{1}{U} \sum_{j=1}^U (T_{c,\hat{i}} + T_{\hat{i},j} - T_{i,j}) \quad (28)$$

where U is again the total number of users. In this work, our goal is to maximize the average transmission delay reduction, and the caching problem is formulated as follows:

$$\mathbf{P3}: \quad \underset{\Phi}{\text{Maximize}} \quad \Delta D \quad (29)$$

$$\text{Subject to} \quad \xi_{i,j} = 1 \quad \exists i \forall j \quad (30)$$

$$\sum_{f=1}^M \phi_{i,f} \leq C_i, \forall i \in \{1, \dots, N\} \quad (31)$$

where Φ is again the $N \times M$ matrix which records the caching states of the N base stations, and each element $\phi_{i,f}$ in the caching state matrix is an indicator, showing if the file is cached at base station i :

$$\phi_{i,f} = \begin{cases} 1 & \text{if the file } f \text{ is cached at the base station } i \\ 0 & \text{if the file } f \text{ is not cached at the base station } i \end{cases} \quad (32)$$

C_i is the maximum number of files that can be stored at base station i . And $\xi_{i,j}$ is an indicator describing if user j is in the area covered by base station i :

$$\xi_{i,j} = \begin{cases} 1 & \text{if user } j \text{ can connect to base station } i \\ 0 & \text{if user } j \text{ cannot connect to base station } i \end{cases} \quad (33)$$

V. DEEP ACTOR-CRITIC FRAMEWORK FOR DECENTRALIZED EDGE CACHING

In this section, we introduce the multi-agent deep reinforcement learning framework for the decentralized edge caching problem. In this framework, there will be multiple DRL agents that can make independent caching decisions based on their own observations, and each agent will also incorporate the Wolpertinger architecture as introduced in subsection III-B.

A. Related Definitions

1) *Agents' Observation and State Space*: Allowing the agents to make their own caching decisions and cooperate with each other, the decentralized caching framework is proposed as a centralized critic network together with a decentralized actor network. Therefore, the agent will feed

the actor network with their own observations and feed the critic network with the complete state space. This multi-agent actor critic framework is based on a partially observable Markov decision process. Each agent i , $i = 1, 2, \dots, N$, can only observe the requests arriving at itself, and select its own action only based on the observation o_i . In the environment, the agent i can observe the contents' features through its local request history. And for the centralized critic, the state space is defined as $\mathbf{x} = \{o_1, o_2, \dots, o_N\}$. Similar to the centralized caching agent's observation, the observation o_i of each decentralized caching agent can be denoted as $o_i = \{\mathcal{F}_s; \mathcal{F}_m; \mathcal{F}_l\}$, where $\{\mathcal{F}_s; \mathcal{F}_m; \mathcal{F}_l\}$ is the feature space as introduced in subsection III-A1.

2) *Action Space*: Similar to the action space in the centralized edge caching framework, introduced in subsection III-A2, each agent in the decentralized caching framework can either keep the current caching state unchanged or only make one replacement. However, in the decentralized caching system, different numbers of file requests arrive at different base stations, since each base station serves potentially different number of users. Therefore, the action space size of each decentralized agent is also different depending on the number of users in the base stations' service regions.

3) *Reward*: The decentralized caching agents are designed to solve the optimization problems **P2** or **P3**, depending on whether cache hit rate or delay reduction is the ultimate goal.

To solve problem **P2**, in operation cycle t , after the agents update their caches according to the selected actions, the cache hit rate for requests in the next operation cycle $t+1$ will be received as the reward within the multi-agent framework. So we define the reward in the t^{th} operation cycle as

$$r_t = P_{hit}^{t+1}, \quad (34)$$

and to solve the problem **P3**, the reward for each iteration is defined as

$$r_t = \Delta D^{t+1}. \quad (35)$$

B. Multi-Agent Actor-Critic Framework

Now, we introduce the decentralized caching framework in detail. Specifically, we have multi-agent actor-critic framework based on the partially observable Markov decision processes with N agents, where the critic network $V(\mathbf{x})$ and N actors $\pi_{\theta_i}(o_i)$, $i = 1, 2, \dots, N$, are parameterized by $\theta = \{\theta_c, \theta_1, \theta_2, \dots, \theta_N\}$.

Actor: The actor network is defined as a function to seek a caching policy $\pi = \{\pi_1, \pi_2, \dots, \pi_N\}$, which can map the observation of the agent to a valid action chosen from the action space \mathcal{A} . In each time slot, agent i will select an action a_i based on its own observation o_i and policy π_i :

$$a_i = \pi_i(o_i). \quad (36)$$

Critic: The critic is employed to estimate the value function $V(\mathbf{x})$, where \mathbf{x} stands for the observation of all agents, $\mathbf{x} = \{o_1, o_2, \dots, o_N\}$. At time instant t , after the actions

$a_t = \{a_{1,t}, \dots, a_{N,t}\}$ are chosen by the actor networks, the agents will execute the actions in the environment and send the current observation \mathbf{x}_t along with the feedback from the environment to the critic. The feedback includes the reward r_t and the next time instant observation \mathbf{x}_{t+1} . Then, the critic can calculate the TD (Temporal Difference) error:

$$\delta^{\pi_\theta} = r_t + \gamma V(\mathbf{x}_{t+1}) - V(\mathbf{x}_t) \quad (37)$$

where $\gamma \in (0, 1)$ is the discount factor.

Update: Instead of using DDPG to train the neural networks as we present in the centralized caching framework, the decentralized caching agents are updated using TD error since this approach involves relatively lower computational complexity, helping the decentralized caching agents more easily meet real-time operation requirements.

Specifically, the critic is updated by minimizing the least squares temporal difference (LSTD):

$$V^* = \arg \min_V (\delta^{\pi_\theta})^2 \quad (38)$$

where V^* denotes the optimal value function.

The actor i is updated by policy gradient. Here, we use TD error to compute the policy gradient:

$$\nabla_{\theta_i} J(\theta_i) = E_{\pi_{\theta_i}} [\nabla_{\theta_i} \log \pi_{\theta_i}(o_i, a_i) \delta^{\pi_\theta}] \quad (39)$$

where $\pi_{\theta_i}(o_i, a_i)$ denotes the score of action a_i under the current policy. Then the weighted difference of parameters in the actor i can be denoted as $\Delta \theta_i = \alpha \nabla_{\theta_i} \log \pi_{\theta_i}(o_i, a_i) \delta^{\pi_\theta}$, where $\alpha \in (0, 1)$ is the learning rate. And the actor network i can be updated using the gradient decent method:

$$\theta_i \leftarrow \theta_i + \alpha \nabla_{\theta_i} \log \pi_{\theta_i}(o_i, a_i) \delta^{\pi_\theta} \quad (40)$$

Workflow: To make the multi-agent system meet real-time operation requirements, we abandon the memory presented in the centralized agent. Consequently, only the current transition will be used in updating the networks. In each iteration, agent i , $i = 1, 2, \dots, N$, observes the features of users' requests and updates its own cache space. Each actor network will propose a proto-actor $\hat{a}_{i,t}$, and each of these proto-actors will be expanded to a K -action set respectively by the KNN, and the expanded action set is denoted as $\mathcal{A}_{i,K}$. Then, one action will be chosen from the expanded action set of each agent, and the chosen actions will be combined to form a new action set to be evaluated by the critic network. Each possible action set combination can be expressed as $\mathbf{A}_h = (\hat{a}_{1,K}, \hat{a}_{2,K}, \dots, \hat{a}_{N,K})$, where we denote $\hat{a}_{i,K}$ as an element chosen from the i^{th} agent's expanded action set. Therefore, there will overall be K^N possible action combinations considering all agents, and we can also index the possible action combination \mathbf{A}_h with $h = 1, 2, \dots, K^N$. The critic network will evaluate all K^N possible combinations of action sets. For example, if there are 2 agents, and each agent's proto-actor is expanded to an action set with 3 actions, the critic network will need to evaluate all 3^2 possible combinations. Following this, the action combination that

provides the maximum state value will be executed in the environment finally. Then, the critic and actor networks will update their parameters accordingly.

The complete process is presented in Algorithm 2 below.

Algorithm 2 Multi-Agent Actor-Critic Algorithm for Content Caching

```

1: Initialize critic network  $V(\mathbf{x})$  and actor  $\pi_{\theta_i}(o_i)$ , parameterized by  $\theta = \{\theta_c, \theta_1, \theta_2, \dots, \theta_N\}$ .
2: Receive initial state  $\mathbf{x} = \{o_1, o_2, \dots, o_N\}$ .
3: for  $t = 1, T$  do
4:   The base station receive users' requests  $Req_t = \{req_{1,t}, req_{2,t}, \dots, req_{U,t}\}$ .
5:   Extract observation at time  $t$  for each agent, and  $\mathbf{x}_t = \{o_{1,t}, o_{2,t}, \dots, o_{N,t}\}$ 
6:   for  $i = 1, N$  do
7:     The agent  $i$  selects proto-actor  $\hat{a}_{i,t} = \pi_{\theta_i}(o_{i,t})$  w.r.t. the current policy
8:     Expand the proto-actor  $\hat{a}_{i,t}$  to a action set with  $k$  actions  $\mathcal{A}_{i,k}$  via KNN.
9:   end for
10:  Extract all possible action set combinations  $\mathbf{A}_h, h = 1, 2, \dots, k^N$ .
11:  Critic network calculate the state value  $V$  with all possible action set combinations.
12:  Find the action set combination that can provide maximum state value, set  $a_t = \arg \max_{\hat{a}_{i,k} \in \mathcal{A}_{i,k}} V(\mathbf{x}_{t+1}|\mathbf{A})$ 
13:  Execute actions  $a_t$  to update the cache state of each base station
14:  Observe reward  $r_t$  and new state  $\mathbf{x}_{t+1}$ 
15:  Critic calculates the TD error based on the current parameter:  $\delta^{\pi_\theta} = r_t + \gamma V(\mathbf{x}_{t+1}) - V(\mathbf{x}_t)$ 
16:  Update the critic parameter  $\theta_c$  by minimizing the loss:  $\mathcal{L}(\theta) = (\delta^{\pi_\theta})^2$ 
17:  for agent  $i = 1$  to  $N$  do
18:    Update the actor policy by maximizing the action value:  $\Delta\theta_i = \alpha \nabla_{\theta_i} \log \pi_{\theta_i}(o_{i,t}, a_i) \delta^{\pi_\theta}, \alpha \in (0, 1)$ .
19:  end for
20:  Update features space  $\mathcal{F}$ 
21: end for

```

VI. NUMERICAL RESULTS

To analyze the performance of our algorithms, comparisons are made between our proposed deep reinforcement learning framework and the following caching algorithms:

- **Least Recently Used (LRU) [33]:** In this policy, the system keeps track of the most recent requests for every cached content. And when the cache storage is full, the cached content, which is requested least recently, will be replaced by the new content.
- **Least Frequently Used (LFU) [34]:** In this policy, the system keeps track of the number of requests for every cached content. And when the cache storage is full, the

cached content, which is requested the least many times, will be replaced by the new content.

- **First In First Out (FIFO) [35]:** In this policy, the system, for each cached content, records the time when the content is cached. And when the cache storage is full, the cached content, which was stored earliest, will be replaced by the new content.

A. Numerical Results for Centralized Edge Caching

1) *Neural Network:* In our implementation, the actor network has two hidden layers of fully-connected units with 256 and 128 neurons, respectively; and the critic network has two hidden layers of fully-connected units with 64 and 32 neurons, respectively. The memory capacity $N_{\mathcal{M}}$ is set as $N_{\mathcal{M}} = 10000$, and the mini batch size is set as $N_{\mathcal{B}} = 100$. The discount factor γ introduced in (12) is set as 0.9. In the KNN component of the algorithm, we conduct the experiments with the number of neighbors as $K_1 = \lceil 0.15C \rceil$ and $K_2 = \lceil 0.05C \rceil$, where C is the cache capacity.

2) *File/Content Request Generation:* In our simulations, the raw data of users' requests is generated according to the Zipf distribution

$$f(k; \beta, M) = \frac{1/k^\beta}{\sum_{m=1}^M (1/m^\beta)} \quad (41)$$

where k is the rank of the files. For each experiment, we collect 10000 requests as the testing data. The settings of Zipf exponent β and total number of files M are specified before each experiment.

3) *Feature Extraction:* From the raw data of content requests, we extract the feature \mathcal{F} and use it as the input state of the network. Here, as features, we consider the number of requests for a file within the most recent 10, 100, and 1000 requests.

Fig. 3 shows the overall cache hit rate (plotted in percentage) achieved by the proposed framework and the other caching algorithms introduced above. In this figure, we set the total number of files as M as 5000 and the Zipf exponent as β at 1.3 and we vary the cache capacity. However, instead of directly using the cache capacity C , we consider the cache ratio $\sigma = \frac{C}{M}$ (where M is the total number of content files that can be requested by the users), so that we can analyze the impact of the cache capacity normalized by the potential data traffic flow into this system. We observe that the proposed framework with $K_1 = \lceil 0.15C \rceil$ outperforms the other strategies and provides the highest cache rates for all cache capacity values, while the performance of the proposed framework with $K_2 = \lceil 0.05C \rceil$ is relatively close to the LFU policy. This observation demonstrates that increasing the number of neighbors selected in the KNN stage can help improve the decision policy because the value of K dictates how many actions can be learned in one iteration. We also notice that when the cache capacity is small, the performance of LFU is very close to our proposed framework. As the cache capacity increases, the gap between proposed framework with

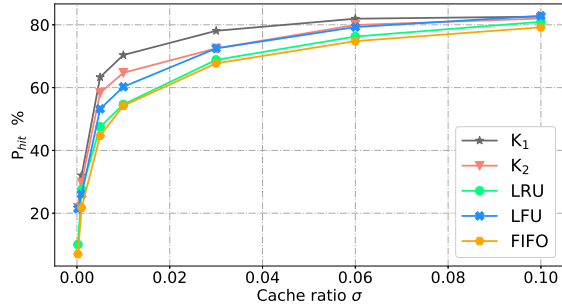


Fig. 3. Cache hit rate vs. cache ratio $\sigma = \frac{C}{M}$. We vary the cache capacity as $C = 1, 5, 25, 50, 150, 300, 500$.

K_1 and other three caching algorithms increases at first, and then gradually decreases. At cache capacity $C = 500$, the cache hit rate of all four algorithms are close to each other at around 80% hit rate. At this point, the cache hit rates achieved by different policies tend to converge because the cache capacity is high enough to store all popular contents. From this point on, increasing the cache capacity will not improve the cache hit rate significantly, and the cache hit rate is now essentially limited by the distribution of the content popularity.

In Fig. 4, we study the cache hit rate as a function of the Zipf exponent β . In this experiment, we set the total number of files as 1000, fix the cache capacity at 100, and vary the Zipf exponent β . Again, we test the proposed framework with two different K values, i.e., $K_1 = \lceil 0.15C \rceil$ and $K_2 = \lceil 0.05C \rceil$, and compare the cache hit rate with that achieved by the non-learning based caching policies. As β increases, cache hit rates achieved by all caching policies grow. This is due to the fact that with larger β , there are fewer files with larger request probabilities and therefore the popularity of the files is skewed. Consequently, caching these more popular files leads to an increase in the cache hit rates. And with the same cache capacity, chances are higher that the agent can cache all the highly popular files (the number of which has decreased). We also notice that the slopes of all the curves first increase and then decrease. This is because initially when the number of popular files gets small, all caching policies start storing the most popular files, and the larger the β , the smaller the influence of less popular files is. However, we eventually experience diminishing returns as β is further increased. In addition, the gap between the curves corresponding to K_1 and K_2 also increases as β gets larger. This verifies that adopting a larger value of K can be helpful for the agent in exploring different action strategies.

In Fig. 5, we plot the cache hit rate as a function of the total numbers of files M . In this experiment, we set the cache capacity as 100, fix the Zipf exponent at 1.4, and vary the value of M . Again the proposed framework is tested with two different values of K and the performance is compared with the LRU, LFU, FIFO caching policies. As the number

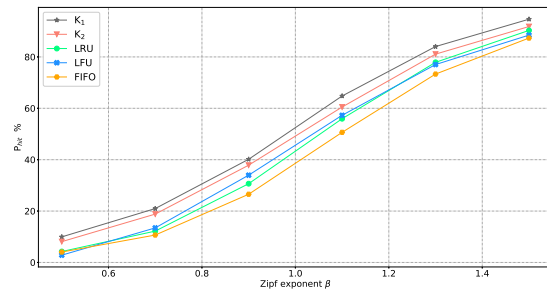


Fig. 4. Cache hit rate vs. Zipf exponent β . We vary the Zipf exponent as $\beta = 0.5, 0.7, 0.9, 1.1, 1.3, 1.5$.

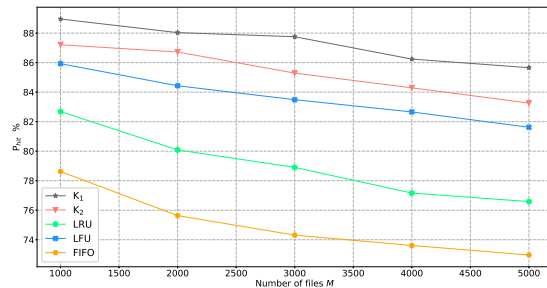


Fig. 5. Cache hit rate vs. number of files M . We vary the number of files as $M = 1000, 2000, 3000, 4000, 5000$.

of files increases, the cache hit rate achieved by all policies tend to decrease. This is because when the cache capacity is fixed, increasing the number of files leads to smaller cache ratio. And since the Zipf exponent β is also fixed, the number of popular files is increased. In this figure, we still observe the proposed framework outperforming for all values of M . Besides, observing the difference between the cache hit rate achieved at $M = 1000$ and that achieved at $M = 5000$ with the same caching policy, we find that the proposed framework has better ability in handling cases with larger M (or smaller cache ratio), making it more competitive in practical settings.

In Fig. 6, we plot the long term average cache hit rate $\bar{P}_{hit}(T) = \sum_{t=0}^T \mathbf{1}(R_t)$ over time. In this experiment, the number of files is fixed at 1000, and the popularities of the files change every 10000 time slots. Each time the popularity changes, the ranks of the files will vary randomly and the Zipf exponent is randomly generated in the range $[1.0, 1.3]$. Note that the change points and the popularity parameters of the files (both ranks and Zipf exponent) are unknown to the reinforcement learning agent. With the long-term average cache hit rate, we evaluate the ability of the caching policies to maintain a stable performance in the changing environment. We can observe that the proposed framework outperforms all the time and the performance is stable. For the LRU and FIFO caching policies, though the curves are flat and smooth, the cache hit rate is not competitive. And for the LFU policy, the cache hit rate drops quickly after the first change point because of the frequency pollution. With this experiment, we conclude that the proposed framework

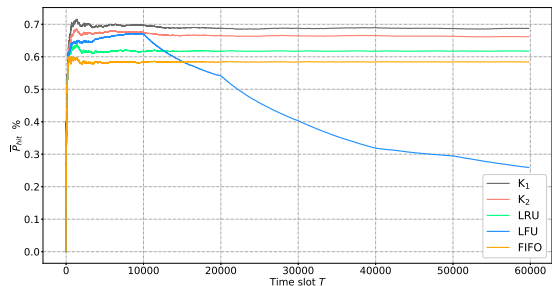


Fig. 6. Long term average cache hit rate as the popularity of files change over time.

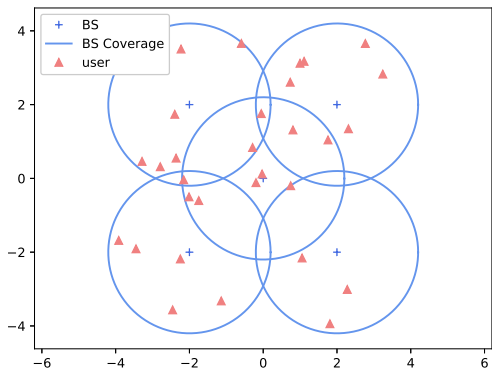


Fig. 7. Coverage map of a system contains 5 base stations and 30 users

is more suitable for applications that require high long-term performance and stability.

B. Numerical Results for Decentralized Edge Caching

1) *Environment Settings*: As shown in Fig. 7, in the experiments, we consider a system with 5 base stations and 30 users randomly distributed in the area, each covered by at least one of the base stations. The cell radius is set as $R = 2.2\text{km}$, and the transmission power of all base stations is set as $P_i = 16.9\text{dB}$, $i = 1, 2, \dots, 5$. The transmission power of the cloud data center is set as $P_c = 20\text{dB}$. As assumed, the content files are split into units of the same size, and the size of each unit is set as 96 bits. And we assume Rayleigh fading with path loss $\mathbb{E}\{z\} = d^{-4}$, where d is the distance between the transmitter and receiver.

2) *Neural Network*: In the implementation, each actor network has three layers, and the first and hidden layers have 200 and 600 neurons, respectively, as shown in Table I. And for each actor network, the number of neurons in the output layer depends on the size of the action space \mathcal{A}_i . For the critic network, the number of neurons in each layer is provided in Table I. To ensure that the critic network can learn faster than the actor networks, we set the learning rate of the critic network as 0.001, and the learning rate of each actor network as 0.0005. And we test this framework with the number of

TABLE I
ARCHITECTURE OF MULTI-AGENT FRAMEWORK

	Actor Network	Critic Network
Input Layer	200 Neurons	400 Neurons
Hidden Layer	600 Neurons	800 Neurons
Output Layer	$\mathcal{D}_i + 1$ Neurons	1 Neurons
Learning Rate	0.0005	0.001

neighbors set as $K = 1$ and $K = 2$ in the KNN component of the algorithm. When $K = 2$, the proto-actor of each agent will be expanded to an action set with 2 valid actions, while when $K = 1$, the proto-actor will not be expanded and the Wolpertinger architecture will specialize to the regular actor-critic structure.

3) *File/Content Request Generation*: In our simulations, the raw data of users' requests is generated according to the Zipf distribution as shown in (41), where the total number of files M is set as 500, and unless state otherwise, the Zipf exponent β is fixed at 1.3 in the study of the cache size and transmission delay. The rank of the file k is randomly generated for each user so that users' preferences for files can be differentiated. To encourage the base station to cache the files that are popular for more users, the users are randomly divided into 5 groups. It is assumed that the users in the same group will have similar but not exactly the same rank for all files. And the group information will not influence the users' location. It is important to note that we generate the requests with Zipf distribution and also group the users. However, such information is totally unknown to the agents.

4) *Feature Extraction*: From the raw data of content requests, we extract the feature \mathcal{F} and use it as the agents' observations of the network. Here, as features, we consider the number of requests for a file within the most recent 10, 100, and 1000 requests.

Cache Hit Rate

In Fig. 8, we plot the overall cache hit rate (as a percentage) achieved by the proposed framework and the other caching policies in a multi-cell network. The tendency of the cache hit rate as the cache ratio increases is very similar to that in the case of a single base station as shown in Fig. 3. However, in a multi-cell scenario, even the cache hit rate achieved by the regular actor-critic framework (i.e., when $K = 1$, whose curves are labeled as "DRL" in the figures) is always higher than those of the LRU, LFU and FIFO policies, while in the single base station case, when we set the number of neighbors as $K_2 = \lceil 0.05C \rceil$, the cache hit rate achieved by the proposed framework can become slightly lower than that of the LFU policy. This observation indicates the benefits of the agents cooperating with each other to avoid caching the same files so that the limited cache storage can be utilized more effectively. Also, when we increase the number of neighbor to $K = 2$ (whose curves are labeled as "K = 2" in the figures), the gap between the

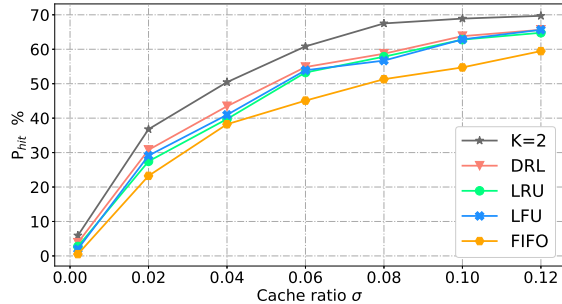


Fig. 8. Cache hit rate vs. cache capacity. We vary the cache capacity as $C = 1, 10, 20, 30, 40, 50, 60$.

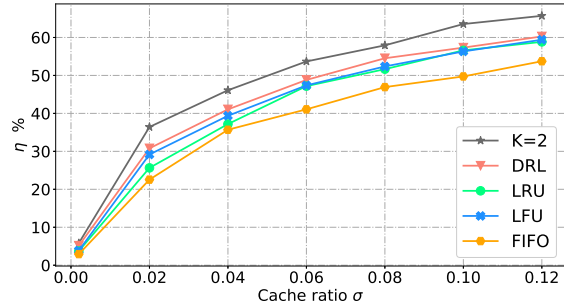


Fig. 10. Percentage of transmission delay reduction vs. cache capacity. We vary the cache capacity as $C = 1, 10, 20, 30, 40, 50, 60$.

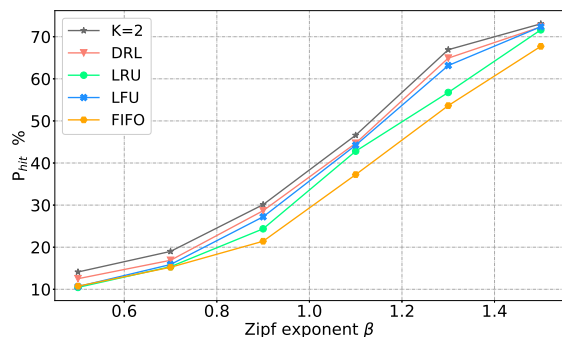


Fig. 9. Cache hit rate vs. Zipf exponent. We vary the Zipf exponent as $\beta = 0.5, 0.7, 0.9, 1.1, 1.3, 1.5$.

proposed framework and other policies increases further. This is because when we increase the number of neighbors from $K = 1$ to $K = 2$, the total number of action sets that will be learned by the critic increases from 1 to 2^5 , which will provide the critic more information for final action selection. Note that we can continue increasing the value of K , but since the number of action sets will increase exponentially, we need to trade off between the performance and increased computational complexity and runtime.

In Fig. 9, we study the relationship between cache hit rate and Zipf exponent β . In this experiment, we fix the cache capacity of all base stations as $C = 40$, and vary the Zipf exponent. As β increases, the cache hit rate of all policies increase because there are now a smaller number of popular files but with higher popularities compared to before when β was smaller. Eventually, for large values of β , performances of different policies tend to converge. This is because with the increasing value of β , the frequencies of popular files being requested is sufficiently high, so that the LFU and LRU policies can always keep these files, and for the proposed learning agents, the features of these files become more distinguishable to learn.

Transmission Delay

In this section, we present the the simulation results addressing the transmission delay. In particular, we evaluate the reduction in transmission delay as a percentage as follows:

$$\eta = \frac{\Delta D}{\frac{1}{U} \sum_{j=1}^U \hat{D}_j} \times 100\%. \quad (42)$$

Hence, η is the percentage of delay reduction per user in one operation cycle.

To determine the relationship between the transmission delay and cache capacity, in Fig. 10, we fix the Zipf exponent at $\beta = 1.3$, and plot the percentage of overall transmission delay reduction η as a function of the cache ratio. It is shown that as the cache ratio σ increases, the reduction in transmission delay achieved by all caching policies first rises quickly because the base stations can cache more files, and then the trend slows down after a certain value of σ . The upward trend starts to slow down because all these caching algorithms are encouraged to cache the most popular files following the statistics they learn. So when the cache ratio grows further and further, the caching agent will start caching the less popular content files. Though more files are cached and transmission delay is further reduced, caching the less popular files at the edge nodes lead to smaller improvements in reducing the transmission delay when compared with the contribution made by caching the most popular files. In other words, when the cache ratio is large enough to cache all of the most popular files, the system does not necessarily have to keep enlarging the cache capacity, considering the price to pay for the storage and the relatively small reduction in transmission delay that will be achieved by storing the less popular files. We also observe that for all values of the cache ratio, the proposed framework achieves better performance for two reasons: First, the proposed framework considers the reduction in the average transmission delay as the reward, so that the caching algorithm does not only focus on finding the most popular files, but also takes into account the users' locations and several less popular files with potentially high delay penalties if not cached; and secondly, the critic network

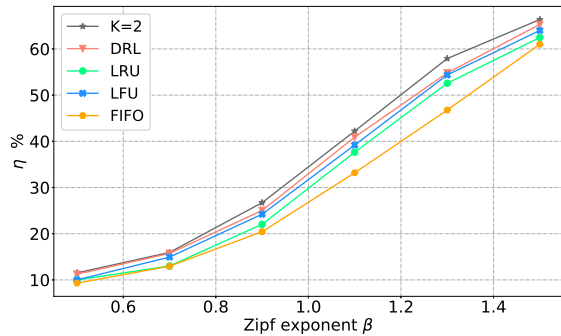


Fig. 11. Percentage of transmission delay reduction vs. Zipf exponent. We vary the Zipf exponent as $\beta = 0.5, 0.7, 0.9, 1.1, 1.3, 1.5$.

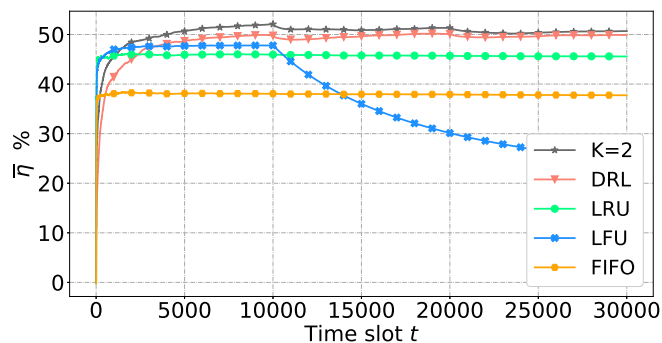


Fig. 12. Percentage of transmission delay reduction η as the popularity distribution of contents change over time

can facilitate the exchange of information among the base stations so that they can avoid caching the same file to serve the user located in overlapped regions, and in this way, utilize the cache space more efficiently.

Again, we fix the cache ratio at $\sigma = 0.1$ and demonstrate how the percentage of transmission delay reduction varies as the Zipf exponent β increases. In Fig. 11, we observe that when β is small, the gap between the curve with “ $K = 2$ ” and the curve labeled “DRL” (i.e., $K = 1$) is small, implying that when the popularities of the files are close to each other, the actor-critic agent is not able to take advantage of the KNN because the features for all files are relatively similar and therefore it is more difficult to find the most popular files. On the other hand, as β increases, the increasing gap between these two curves points to the advantage of adopting a larger number of neighbors in KNN. And when the value of β approaches 1.5, the actor-critic agents with different K values achieve similar performances again since the features of popular files can be easily distinguished from the non-popular files, and therefore even with smaller number of neighbors, the proposed framework can learn it well.

In Fig. 12, we demonstrate the ability of the caching policies to adapt to varying content popularity distributions. In this experiment, the users’ preferences for files change at every 10000 time slots. The users’ requests are generated

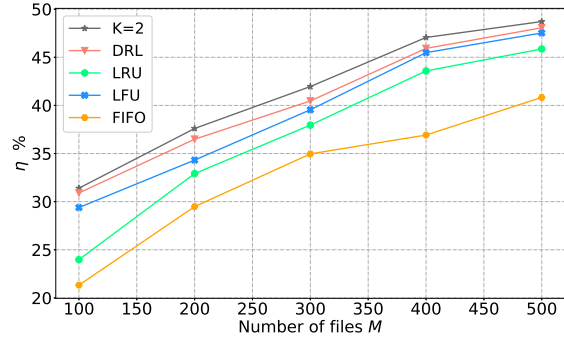


Fig. 13. Percentage of transmission delay reduction vs. total number of files. We fix the Zipf exponent at $\beta = 1.3$ and the cache ratio $\sigma = 0.1$.

using Zipf distributions with their unique ranks of files and Zipf exponents. At each change point, these parameters vary randomly. The change points and Zipf parameters are all unknown to the caching agents. We only limit the Zipf exponent β to be in the range $[1.1, 1.5]$. Then we plot the average of the percentages of the transmission delay reduction over time as $\bar{\eta}_T = \frac{1}{T} \sum_{t=1}^T \eta_t$, for $t = 1, 2, \dots, 30000$. As shown in Fig. 12, the proposed framework with both values of number of neighbors achieve relatively lower performance at the beginning, because unlike the other three caching policies (i.e., LRU, LFU, and FIFO), the proposed framework does not directly collect the statistics from the users’ requests, but generally adjust the parameters of the neural networks and learn the popularity patterns of the files. After the neural networks are trained well, the two actor-critic agents are able to achieve better long-term performance over the other policies. As before, having larger number of neighbors (i.e., $K = 2$) results in the best performance. And at each time the popularity distribution changes, even though the average transmission delay reduction slightly drops as the actor-critic framework updates the parameters to adapt to the new pattern, the actor-critic agents can keep a stable performance after the re-training process. The LFU policy performs the best at the beginning, but due to the frequency pollution, the performance drops quickly at the first change point and continues diminishing. For the LRU and FIFO policies, the performances are stable, because the cache size is limited and the files that are used to be popular and less popular after the change can be replaced in a relatively short amount of time. However, as evidenced in this figure, their performances are lower and the proposed framework is more suitable to be applied in scenarios that require long-term high performance and stability.

In Fig. 13, we plot the percentage of transmission delay reduction as a function of the total number of files. Actor-critic agents again outperform the other caching policies.

In this work, we have focused on edge caching in single-cell and multi-cell scenarios. In particular, we have designed deep actor-critic reinforcement learning frameworks for both centralized and decentralized edge caching scenarios. More specifically, we have employed the Wolpertinger architecture involving an actor neural network, a KNN component, and a critic neural network. We have described in detail how the neural networks are updated. We have developed a single-agent actor-critic algorithm in the single-cell scenario and described its workflow. In the multi-cell setting, we have proposed a decentralized edge caching strategy via a multi-agent framework with multiple actor networks and a single critic network. In this setting, we have designed a multi-agent actor-critic algorithm. We have provided simulation results to test the performance of the proposed frameworks. For the centralized edge caching scenario, we have analyzed the performance in terms of the cache hit rate as a function of the cache ratio, Zipf exponent, and the number of files. For decentralized edge caching in a multi-cell environment, we have considered two objectives: cache hit rate and transmission delay reduction. We have studied the performance in terms of both objectives again as the cache ratio, Zipf exponent, and the number of files vary. We have also evaluated the reinforcement learning agents' adaptation capabilities in the presence of unknown change points where users' preferences change randomly. In all of the experiments, the proposed actor-critic frameworks have shown advantage over the non-learning based policies, leading to benefits and improvements in terms of cache hit rate, transmission delay reduction, adaptation capability and long-term stability.

REFERENCES

- [1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update, 2017-2022 white paper," Available: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html>, [Online], Feb. 2019.
- [2] Q. Yan, M. Cheng, X. Tang, and Q. Chen, "On the placement delivery array design for centralized coded caching scheme," *IEEE Transactions on Information Theory*, vol. 63, no. 9, pp. 5821–5833, 2017.
- [3] Q. Yang, P. Hassanzadeh, D. Gündüz, and E. Erkip, "Centralized caching and delivery of correlated contents over a Gaussian broadcast channel," in *Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2018 16th International Symposium on, pp. 1–6, IEEE, 2018.
- [4] K. Kvaternik, J. Llorca, D. Kilper, and L. Pavel, "A methodology for the design of self-optimizing, decentralized content-caching strategies," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2634–2647, 2016.
- [5] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 8, pp. 1791–1805, 2018.
- [6] Y. Li, C. Zhong, M. C. Gursoy, and S. Velipasalar, "Learning-based delay-aware caching in wireless D2D caching networks," *IEEE Access*, vol. 6, pp. 77250–77264, 2018.
- [7] W. Wang, D. Niyato, P. Wang, and A. Leshem, "Decentralized caching for content delivery based on blockchain: A game theoretic perspective," *arXiv preprint arXiv:1801.07604*, 2018.
- [8] B. Zhou, Y. Cui, and M. Tao, "Optimal dynamic multicast scheduling for cache-enabled content-centric wireless networks," *IEEE Transactions on Communications*, vol. 65, no. 7, pp. 2956–2970, 2017.
- [9] J. Kwak, Y. Kim, L. B. Le, and S. Chong, "Hybrid content caching in 5G wireless networks: Cloud versus edge caching," *IEEE Transactions on Wireless Communications*, vol. 17, no. 5, pp. 3030–3045, 2018.
- [10] M. Chen, W. Saad, C. Yin, and M. Debbah, "Echo state networks for proactive caching in cloud-based radio access networks with mobile users," *IEEE Transactions on Wireless Communications*, vol. 16, no. 6, pp. 3520–3535, 2017.
- [11] T. X. Tran, A. Hajisami, and D. Pompili, "Cooperative hierarchical caching in 5G cloud radio access networks," *IEEE Network*, vol. 31, no. 4, pp. 35–41, 2017.
- [12] X. Li, X. Wang, P.-J. Wan, Z. Han, and V. C. Leung, "Hierarchical edge caching in device-to-device aided mobile networks: Modeling, optimization, and design," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1768–1785, 2018.
- [13] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, "Placing dynamic content in caches with small population," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pp. 1–9, IEEE, 2016.
- [14] W. Li, S. M. Oteafy, and H. S. Hassanein, "Streamcache: popularity-based caching for adaptive streaming over information-centric networks," in *Communications (ICC), 2016 IEEE International Conference on*, pp. 1–6, IEEE, 2016.
- [15] S. Li, J. Xu, M. Van Der Schaar, and W. Li, "Popularity-driven content caching," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pp. 1–9, IEEE, 2016.
- [16] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.
- [17] R. Pedarsani, M. A. Maddah-Ali, and U. Niesen, "Online coded caching," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 836–845, 2016.
- [18] J. Song and W. Choi, "Mobility-aware content placement for device-to-device caching systems," *IEEE Transactions on Wireless Communications*, vol. 18, pp. 3658–3668, July 2019.
- [19] M. S. ElBamby, M. Bennis, W. Saad, and M. Latva-Aho, "Content-aware user clustering and caching in wireless small cell networks," in *2014 11th International Symposium on Wireless Communications Systems (ISWCS)*, pp. 945–949, IEEE, 2014.
- [20] 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, 2018.
- [21] L. Lei, L. You, G. Dai, T. X. Vu, D. Yuan, and S. Chatzinotas, "A deep learning approach for optimizing content delivering in cache-enabled HetNet," in *Wireless Communication Systems (ISWCS), 2017 International Symposium on*, pp. 449–453, IEEE, 2017.
- [22] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5G using reinforcement learning of space-time popularities," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 180–190, 2018.
- [23] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *Information Sciences and Systems (CISS), 2018 52nd Annual Conference on*, pp. 1–6, IEEE, 2018.
- [24] Y. Wei, F. R. Yu, M. Song, and Z. Han, "Joint optimization of caching, computing, and radio resources for fog-enabled iot using natural actor-critic deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2061–2073, 2018.
- [25] J. Sung, K. Kim, J. Kim, and J.-K. K. Rhee, "Efficient content replacement in wireless content delivery network with cooperative caching," in *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*, pp. 547–552, IEEE, 2016.
- [26] J. Song, M. Sheng, T. Q. Quek, C. Xu, and X. Wang, "Learning-based content caching and sharing for wireless networks," *IEEE Transactions on Communications*, vol. 65, no. 10, pp. 4309–4324, 2017.
- [27] A. Sengupta, S. Amuru, R. Tandon, R. M. Buehrer, and T. C. Clancy, "Learning distributed caching strategies in small cell networks," in *Wireless Communications Systems (ISWCS), 2014 11th International Symposium on*, pp. 917–921, IEEE, 2014.

- [28] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.
- [29] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, pp. 6382–6393, 2017.
- [30] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 2137–2145, 2016.
- [31] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*, pp. 66–83, Springer, 2017.
- [32] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
- [33] M. Ahmed, S. Traverso, P. Giaccone, E. Leonardi, and S. Niccolini, "Analyzing the performance of LRU caches under non-stationary traffic patterns," *arXiv preprint arXiv:1301.4909*, 2013.
- [34] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *ACM SIGARCH Computer Architecture News*, vol. 38, pp. 60–71, ACM, 2010.
- [35] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, pp. 1–6, 2011.



Chen Zhong is currently a Ph.D. student in the Department of Electrical Engineering and Computer Science at Syracuse University. She received her B.S. degree in Information Engineering from Beijing Institute of Technology (China) in 2014 and her M.S. degree in Electrical Engineering from Stevens Institute of Technology in 2016. Her research interests are in the areas of wireless communication and networking. Currently, she is working on content caching and resource management problems using the tools from machine

learning and optimization.



M. Cenk Gursoy received the B.S. degree with high distinction in electrical and electronics engineering from Bogazici University, Istanbul, Turkey, in 1999 and the Ph.D. degree in electrical engineering from Princeton University, Princeton, NJ, in 2004. He was a recipient of the Gordon Wu Graduate Fellowship from Princeton University between 1999 and 2003. Between 2004 and 2011, he was a faculty member in the Department of Electrical Engineering at the University of Nebraska-Lincoln (UNL). He is currently a Professor in the

Department of Electrical Engineering and Computer Science at Syracuse University. His research interests are in the general areas of wireless communications, information theory, communication networks, signal processing, and machine learning. He is currently a member of the editorial boards of IEEE Transactions on Wireless Communications and IEEE Transactions on Green Communications and Networking, and he is an Area Editor for IEEE Transactions on Vehicular Technology. He also served as an editor for IEEE Transactions on Wireless Communications between 2010 and 2015, IEEE Communications Letters between 2012 and 2014, IEEE Journal on Selected Areas in Communications - Series on Green Communications and Networking (JSAC-SGCN) between 2015 and 2016, Physical Communication (Elsevier) between 2010 and 2017, and IEEE Transactions on Communications between 2013 and 2018. He has been the co-chair of the 2017 International Conference on Computing, Networking and Communications (ICNC) - Communication QoS and System Modeling Symposium, the co-chair of 2019 IEEE Global Communications Conference (GlobeCom) - Wireless Communications Symposium, and the co-chair of 2019 IEEE Vehicular Technology Conference Fall - Green Communications and Networks Track. He received an NSF CAREER Award in 2006. More recently, he received the EURASIP Journal of Wireless Communications and Networking Best Paper Award, 2019 The 38th AIAA/IEEE Digital Avionics Systems Conference Best of Session (UTM-4) Award, 2017 IEEE PIMRC Best Paper Award, 2017 IEEE Green Communications & Computing Technical Committee Best Journal Paper Award, UNL College Distinguished Teaching Award, and the Maude Hammond Fling Faculty Research Fellowship. He is a Senior Member of IEEE, and is the Aerospace/Communications/Signal Processing Chapter Co-Chair of IEEE Syracuse Section.



Senem Velipasalar received the B.S. degree (with high honors) in electrical and electronic engineering from Bogazici University, Istanbul, Turkey, in 1999; the M.S. degree in electrical sciences and computer engineering from Brown University, Providence, RI, USA, in 2001; and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, USA, in 2004 and 2007, respectively.

During the summers of 2001–2005, she was with the Exploratory Computer Vision Group, IBM T. J. Watson Research Center, NY, USA. Between 2007 and 2011, she was an Assistant Professor with the Department of Electrical Engineering, University of Nebraska–Lincoln, Lincoln, NE, USA. She is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY. Her research focuses on mobile camera applications, wireless embedded smart cameras, multi-camera tracking and surveillance systems, and automatic event detection from videos. Her research interests include embedded computer vision, video/image processing, embedded smart camera systems, distributed multi-camera systems, pattern recognition, and signal processing.

Dr. Velipasalar received a National Science Foundation CAREER Award in 2011, Excellence in Graduate Education Faculty Recognition Award in 2014, and the Best Student Paper Award at the IEEE International Conference on Multimedia and Expo in 2006. She has also received the EPSCoR First Award, two Layman awards, the IBM Patent Application award, and graduate fellowships from Princeton University and Brown University. She is a member of the Editorial Board of the Springer Journal of Signal Processing Systems.