Multi-Agent Deep Reinforcement Learning for Cooperative Edge Caching via Hybrid Communication

Fei Wang[1], Salma Emara[1], Isidor Kaplan[1], Baochun Li[1], and Timothy Zeyl[2]

[1]Department of Electrical Computer Engineering, University of Toronto
[2]Huawei Canada

***Abstract*—Though caching on edge servers is widely acknowledged to be essential, it is not trivial to cache content on edge servers adaptively without any prior knowledge of the distribution of content popularity across the users. Several edge caching algorithms have been proposed in the literature based on multi-agent reinforcement learning (MARL) for dynamic control, however, they ignored the non-stationarity and partial-observability issues commonly existing in multi-agent systems. In an MARL-based edge caching application where agents collaborate towards a common goal, communication is essential as their decisions are jointly applied to improve collective intelligence. However, most existing methods proposed to exchange messages between agents have not considered the induced communication overhead, which is critical in practice with real-world multi-agent applications. In this paper, we propose a new MARL framework for edge caching where agents learn to construct, exchange and interpret collective messages for individual benefits, while controlling the complex collaborative task of cache replacement in a communication-efficient manner. With a standard edge caching model, we show that with limited communication and delays introduced, our proposed framework is able to outperform existing rule-based and learning-based caching policy alternatives.**

***Index Terms*—Multi-agent reinforcement learning, learning to communicate, edge caching**

I. Introduction

Given the advent of 5G networks, edge caching has been attracting a significant amount of interest in the literature. Content caching [1], [2] has emerged as a technique that allows edge servers that are closer to end users in wireless networks to proactively cache popular data and deliver the requested data to the users. Moreover, edge servers can turn to neighboring edge servers to retrieve the requested data and return them to the users, in which circumstance data traffic is effectively reduced compared to downloading data directly from the data center via backhaul networks.

Existing studies [3]–[5] have shown the great potential for multi-agent reinforcement learning (MARL) to design content caching strategies at the network edge. The proposed approaches in these studies easily outperformed heuristic caching schemes. MacoCache [5] was able to achieve better performance than the state-of-the-art single-agent DRL-based caching solution [6]. With the collaboration of multiple agents towards a common goal, communication becomes more essential as their decisions are jointly applied whereas each agent only has partial or limited observability of the entire environment. However, none of the existing MARL-based edge caching methods considered inter-agent communication during reinforcement learning control.

Numerous studies have demonstrated the effectiveness of learning to communicate in multi-agent collaboration. Naghizadeh *et al.* [7] evaluated the benefits and drawbacks of communication towards both coordination and the learning progress in multi-agent reinforcement learning. In general, these existing multi-agent communication schemes fall into two categories according to when the messages are exchanged: *intra-step* communication (e.g., CommNet [8], ATOC [9], VBC [10], I2C [11]) and *inter-step* communication (e.g., DIAL [12] and TarMAC [13]).

In intra-step communication, agents exchange messages within every time step from observing the environment to applying the next action. Therefore, steps might be extended for agents to send and receive messages before they can derive actions combining local observations and the information in messages from others. This will introduce a considerable delay to the decision making process of the primary reinforcement learning task, as messages are transmitted through the communication network. In inter-step communication, agents' decision making will not be delayed by the message exchanging out of each step. However, these messages exchanged after the previous step's decision making can only incorporate the information or experience from the previous steps but not any other agents' statuses and beliefs in the current step.

To design a practical multi-agent communication strategy, it's necessary to consider both the potential latency induced by intra-step communication and the limited information conveyed by message exchanges in inter-step communication. Though most existing MARL algorithms in the literature considered the case of limited communication bandwidth, their designs were only empirically evaluated in static games and a small grid world with basic simulated physics [14]. Furthermore, few of the existing communication strategies with multi-agent reinforcement learning are sufficiently practical, when it comes to applying them to real-world scenarios. This is because they fail to consider the extra communication delay with real-world communication networks that are often limited in bandwidth availability, especially when there are a large

number of agents. For example, in the edge caching context, the limited-bandwidth restriction can be interpreted as the limited transmission capacity of network links. It remains an open challenge to implement multi-agent communication in real-world decentralized decision making systems, with practical restrictions on transmitting messages among the agents.

Motivated by the need to support MARL in real-world decision-making processes with bandwidth constraints in the underlying communication network, in this paper, we explore the design of a hybrid communication pattern that (1) involves both intra-step and inter-step communication, (2) introduces limited delays during agents' decision making, and (3) incorporates both history and current knowledge into the shared messages. To achieve these objectives, we propose a new multi-agent communication framework for cooperative multi-agent reinforcement learning in edge caching tasks, referred to as *Communicator with Successive Deep Neural Networks*, or CSNet. Our new framework combines specially designed deep neural networks that adaptively convert local knowledge to restricted-sized messages, together with neural networks that selectively integrate shared messages into a piece of global knowledge that is beneficial for the agents' coordination and learning. Agents in our framework are able to learn what information should be shared, how global knowledge should be understood, how history global knowledge should be kept in local memory through a multi-agent deep reinforcement learning process, with the objective of reducing the cost of communication while maintaining the same level of performance. Overall, we aim at improving each agent's knowledge about the dynamic environment while introducing limited delays in agents' decision making.

Our experimental results demonstrate that our communication strategy for multi-agent reinforcement learning, CSNet, assists the efficient coordination across edge servers in wireless edge caching. CSNet outperforms alternative MARL algorithms with no communication [15] or with full observation sharing [7] when they are applied to generating caching policies, in terms of cache hits, transmission latency, and replacement costs. Meanwhile, CSNet can save communication bandwidth up to 97.5% compared to the full observation sharing.

II. Cooperative Edge Caching with Multi-Agent Reinforcement Learning

A. System Model

We consider the edge caching model illustrated in Fig. 1. Each edge server is located at the center of each wireless network region, and is connected to the datacenter. Naturally, we assume that each edge server has a limited caching capacity, and can only cache a portion of the data stored in the datacenter. The users are randomly distributed across the network regions. At a particular time step, each user can request one piece of data based on his or her preference from the local edge server. Users make data requests from a finite data set.

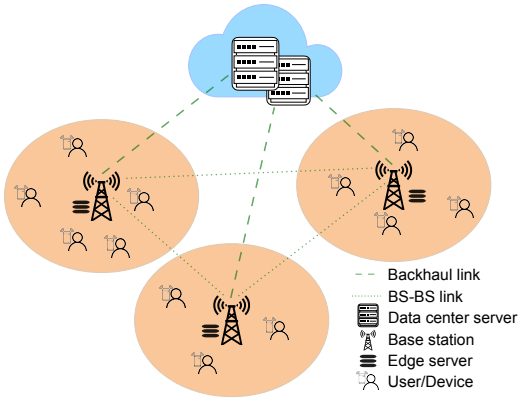


Fig. 1: Cooperative edge caching: system model.

The popularity distribution of data is assumed to be unknown when making caching decisions. Hence, each edge server serves the data requests from users within its coverage area if the requested data can be found in its cache. Otherwise, the edge server can request any locally uncached data from neighboring edge servers. The worst case is that the requested data can not be obtained at either the local cache or the neighboring cache, and the edge server will fetch the data from the datacenter. We assume that the edge server of each user remains unchanged within each time step.

The entire data set $\mathcal{F}$ consists of F pieces of files. We assume each piece of data as file f has a unique ID, and is of the same size s_f. There are E edge servers in the system, among which every edge server i has a cache to store the limited number of files. The cache can be regarded as memory blocks, each block c is the same size as the file, and the total cache size is denoted as C_i. And $\mathcal{N}_i$ denotes the neighboring edge servers of i (located within a certain distance), which can exchange cached data with i. The total number of users distributed across the network regions is U.

Based on the system model presented above, we describe the important metrics that we take into consideration when designing the RL formulation and experimental evaluation. The total hit ratio at an edge server i is calculated as the total number of cache hits, including local and neighbor cache hits, to the total number of cache requests in the given period. Alternatively, the total hit ratio can be represented as the sum of the *local hit ratio* $\mathrm{LocalHR}_i$ and the *neighbor hit ratio* $\mathrm{NeighborHR}_i$. Note that the neighbor cache hits increment if the requested data is not available at the local cache but can be found at any one of the neighbor cache. The *replacement ratio* RR_i is calculated as the ratio of the total number of file replacements occurred in the cache over the total number of files that could be held in the cache in a given period.

According to the Shannon-Hartley theorem, we can calculate the maximum transmission rate r_l on a link l between a user and its associated edge server as $r_l = B_l \cdot \log_2(1+\mathrm{SNR}_l)$, where B is the allocated channel bandwidth of the link. The SNR_l represents the signal-to-noise ratio on the link l, determined by the server's transmission power, the path loss,

and the path distance between the data sender and receiver. Hence, the transmission latency caused by the event that user u requests file f from edge server i can be denoted as $L^f_{l_{u-i}} = s_f / r_{l_{u-i}}$. Similarly, the transmission latency on the other links, *i.e.,* the link between an edge server and its neighbor, can be obtained. We denote the *total transmission latency* at edge server i caused by its response to the data requests during the given period as L_i.

B. Reinforcement Learning Formulation

We treat edge servers as agents who cooperate with each other and update the local caches upon their strategies, and who aim to minimize the overall transmission latency for users' requested data to achieve global optimality. At each discrete time step t, each agent i takes an action a_i^t based on its local observation o_i^t and its policy $\pi_i(a_i|o_i)$ and receives an observation o_i^{t+1} correlated with the new state and a joint reward r^t. During interactions with the environment, agents aim to develop an optimal joint policy $\pi^\star$ mapping observations to actions that maximizes the discounted cumulative reward (*i.e.,* the expected return) $\mathbb{E}[\sum_{t=0}^{h-1} \gamma^t r^t]$, where $h = \infty$ when the number of steps is infinite, and the discount factor $\gamma \in [0, 1]$ describes how important future rewards are for the agents. Accordingly, we form a multi-agent reinforcement learning system based on the partially observable Markov decision processes (POMDPs) formulations [16].

State space. The state space of an agent i consists of the *content caching state* $\mathbf{C}_i$ and the *content requesting state* $\mathbf{Q}_i$ denoted as $\mathbf{S}_i : \{\mathbf{C}_i = (c_{i,1}, c_{i,2}, ..., c_{i,F}), \mathbf{Q}_i = (q_{i,1}, q_{i,2}, ..., q_{i,F})\}$.

In the *content caching state*, we have $c_{i,f} = 1$ if file f is cached at the edge server i and $c_{i,f} = 0$ otherwise. Similarly, each element $q_{i,f}$ in the *content requesting state* indicates whether or not the edge server i has received the request of file f by some end user. At each time step t, the agent i obtains its state $\mathbf{s}_i^t \in \mathbf{S}_i$.

Action space. Each agent has to make an action $\mathbf{a}_i^t$ to replace the files in the local cache during the step t. Our initial design of the action space was $\mathbf{A}_i^t = \{(a_{i,1}^t, a_{i,2}^t, ..., a_{i,F}^t)\}$, where $0 \leq a_{i,f} \leq 1$ represents the probability of caching the file f in the next step. According to the caching capacity C_i of the edge server i, the corresponding number of files with the highest probabilities will be selected to replace the cache.

However, this design could bring a superfluous action space as there is a large amount of files in the content library. To avoid complicating the control of this cooperative edge caching scenario, we alternatively designed the action space as $\mathbf{A}_i^t = \{(a_{i,1}^t, a_{i,2}^t, ..., a_{i,C_i}^t)\}$, where $0 \leq a_{i,c}^t \leq F$ (rounded) represents the ID of the file to be cached in the cache block c. Therefore, the action space is only restricted by the local cache capacity instead.

Reward function. We penalize agents for large *transmission latency* L_i^t, and large *replacement ratio* RR_i^t while rewarding them for high *local hit ratio* $\mathrm{LocalHR}_i^t$ and *neighbor hit ratio* $\mathrm{NeighborHR}_i^t$. Thus, the reward is simply a linear function of these metrics given different weights.

Training. We follow the paradigm of centralized training with decentralized execution [15] for multi-agent control, where during training, the agents have the opportunity to access extra information apart from local observations that are not available during execution. Each agent maintains an individual actor network to access local observations and take actions, while there is a central critic network taking responsibility for the training and is empowered to observe the full experience. This framework can effectively mitigate the common non-stationary issues in multi-agent settings.

In particular, we use the Deep Deterministic Policy Gradient (DDPG) [17] algorithm for learning, which is an improvement over the actor-critic algorithm [18]. The action-value function (or Q function) for the policy π is defined as $Q^\pi(s, a) = \mathbb{E}[R|s^t = s, a^t = a]$. DDPG learns a Q function and a policy in an off-policy way to iterate over actions. Using the Q function defined previously, the objective in the DDPG algorithm is to maximize the expected return written as $J(\theta) = \mathbb{E}[Q(s, a)|s^t = s, a^t = \pi(s)]$. Hence, the gradient can be expressed as

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \boldsymbol{\pi}_\theta(s) \cdot \nabla_a Q^{\boldsymbol{\pi}}(s, a)], \qquad (1)$$

where ρ^π is the state distribution.

In DDPG, the actor $\boldsymbol{\pi}_\theta$ is a policy network that directly maps a state to an exact action in continuous space instead of a probability distribution over actions, and the critic $Q^{\boldsymbol{\pi}}$ is a Q-value network that parameterizes the Q-value by the state and the action. As used in the DQN algorithm [19] and many other reinforcement learning algorithms, DDPG also makes use of a replay buffer of experience and target networks to stabilize the learning behavior.

III. Hybrid Communication for MARL

Communication among agents can make the problem modeled as a decentralized partially observable Markov decision process (Dec-POMDP) easier to solve, as better policy $\pi_i(a_i|o_i, m_i)$ rather than $\pi_i(a_i|o_i)$ is learned by each agent, where m_i denotes the communication message received by agent i. With the objective of maximizing the expected return, agents also learn a protocol for communicating messages π^m apart from the optimal policy π.

An extreme case is that all agents share with each other their local observations so that the problem is simplified as a POMDP with centralization [16]. Hence, each agent receives a message $m_i = \boldsymbol{o}_{-i}$ that consists of the joint observations of other agents except itself. However, communication over real-world networks will induce extra cost or delay in the system, which makes it unrealistic to share every local information during the training. As a result, communication message m_i needs to be carefully designed to optimize the learned policy introducing limited cost or delay. What, when and with whom to communicate will jointly influence the complexity of m_i.

A. CSNet: Communicator with Successive Deep Neural Networks

Our proposed MARL communication framework is illustrated in Fig. 2. By extending the multi-agent deep determin-

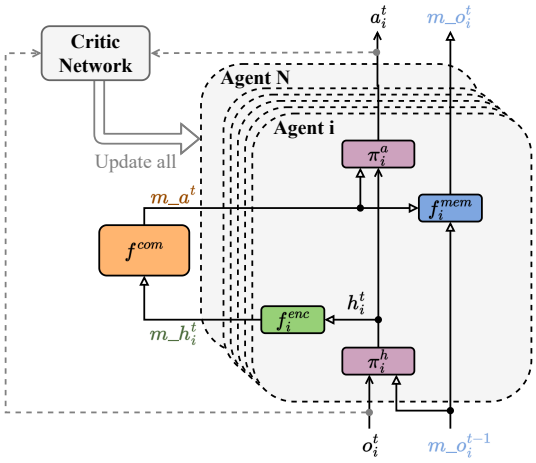


Fig. 2: Multi-agent reinforcement learning with communication.

istic policy gradient (MADDPG) model [15], which will be referred to as the *no communication* scheme, we are able to support efficient message sharing by involving both intra-step and inter-step communication.

Deriving knowledge from the actor. It is essential to decide what information to exchange between multiple agents. Sharing observations and/or behavior policies of all agents with every other agent [5], [7] requires a significant amount of data transmission overhead in real-world applications. Since a large amount of information needs to be aggregated at each agent, the ensuing communication costs could be prohibitive. In addition, informing all the agents of what other agents observe and how they behave could overwhelm each of the agents [20]. This kind of information sharing is not only devoid of value, but also detrimental to the learning process of the entire system, especially for agents who have different capacities of observability or heterogeneous policies.

To transfer knowledge that is not only cheaper to transmit but also easier to digest for agents, we seek a higher-level representation of both local perceptions and action intentions rather than raw observations or behavior policies. In our design, as shown in Fig. 2, information sharing among agents takes place as we generate actions from observations according to current policies. We split the actor network into two parts, in which the first part — the *Actor Head* π_i^h — outputs the hidden state h_i^t that can be seen as a feature of the local observation o_i^t at each time step. The *Actor Head* comprises two fully connected (FC) layers with ReLU as the activation function. We use the hidden state h_i^t as a piece of information that the agent owns locally.

Message compression. As the initial message h_i^t is the immediate output of the first part of the actor neural network, it could still be inefficient to send them out directly through the communication channels. Therefore, to reduce the volume of traffic used by the agents' messages, we compress it by feeding it into a *Message Encoder*, f_i^{enc}. The message encoder is a fully connected layer with parameters θ_i^e, which reduces the dimensionality of the immediate output to a high-level message. Hence, the message to be sent by each agent i at step t can be written as

$$m\_h_i^t = f_i^{enc}(h_i^t). \tag{2}$$

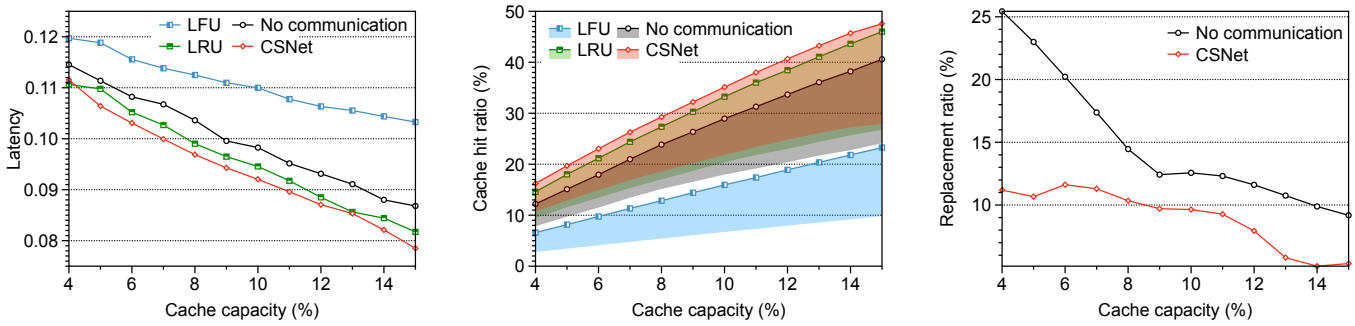
The length of message $m\_h_i^t$ is adjustable by the output size of the encoder. Despite the fact that agents are communicating during execution, the incurred message transmission cost and latency are reduced to the minimum using the message encoder.

Message integration. The agents share their encoded local messages $\{m\_h_1^t, ..., m\_h_N^t\}$ at each time step t through a shared communication channel. The shared information will be integrated into a global message m_a^t, and then broadcast to each agent i. The second part of the actor network at each agent, the *Actor Tail* π_i^a, takes the local hidden states h_i^t and the global message m_a_t as input, and generates the next action a_i^t. The *Actor Tail* is a fully connected layer. The output of the common communicator is given by

$$f^{com}(\mathbf{M}\_h^t) =: f^{com}(m\_h_1^t, m\_h_2^t, ..., m\_h_N^t) = m\_a^t. \tag{3}$$

It is vital to effectively integrate the collected information sent by each agent into a useful global message that is valuable for the coordination and learning of the entire group. Intuitively, we first examined several message integration methods such as taking the average value of all the local message vectors, or simply concatenating the message vectors sequentially. However, these two naïve designs are not able to satisfy our need for adaptively differentiating the values of different messages, and aggregating them with different weights. Pieces of information of high importance may get mitigated, or even worse, there may be redundant or unnecessary information in the messages. For this reason, we further employ a new neural network as an alternative to implement the message integration component, referred to as the *Deep* Communicator. Instead of using a fixed mapping, the *Deep* Communicator adopts a recurrent neural network with parameters θ^c to take in the messages from each agent sequentially, and adaptively generate a broadcast message of a reduced size. It retains its own memory across the training steps and keeps learning what is important to be shared with the agents.

Message memory. The message m_a described above are generated, sent, and received during each single step, from observing the environment to applying the next action, and thus only incorporate information available within each step. Apart from this message fed into the *Actor Tail* that indicates the action intention of the group of agents, our design also involves a self-updating message m_o_i aimed at informing the agents of history knowledge besides the local observation. That is, each agent retains a memory f_i^{mem} of previous global messages from the group of agents, which enables the agent to sense the world more than its next observation. The memory component f_i^{mem} is a fully connected layer with parameters



(a) Latency (b) Total (local + neighbor) hit ratio (c) Replacement ratio

Fig. 3: The latency, cache hit ratio, replacement ratio of different policies under varying cache capacities.

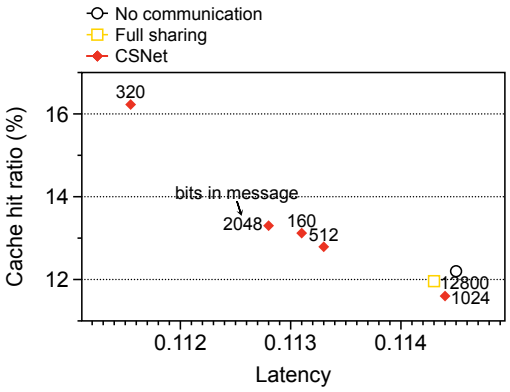


Fig. 4: The average hit ratio and latency of different message size under 4% cache capacity.

θ_i^m. The message $m\_o_i^t$ generated at the end of the step t will be fed into the *Actor Head* along with the new observation o_i^t at the next step $t+1$. The message flow through the memory component can be expressed as

$$m\_o_i^t = f_i^{mem}(m\_a^t, m\_o_i^{t-1}). \tag{4}$$

In our CSNet design, the aggregated actor network, message encoder, message memory and Deep Communicator work together to encode and integrate the local messages m_h, decode the global messages m_a, and produce the actions. Specifically, the gradients flow back through *Actor Tail* π^a, the *Deep* Communicator f^{com}, the message memory f_i^{mem}, the message encoder f_i^{enc}, and *Actor Head* π^h for each agent. As the common message integration component, the *Deep* Communicator is trained jointly by incorporating the partial gradients contributed by each agent. Therefore, the gradient in Eq. (1) can be extended as Eq. (5), and the gradient for updating parameters in the Deep Communicator can be expressed as Eq. (6), where the state distribution ρ^π is derived from the replay buffer.

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{o}\sim\rho^\pi, \boldsymbol{a}\sim\pi_{\theta_i}}[\nabla_{\theta_i}\boldsymbol{\pi}_{\theta_i}(o_i, M)\cdot\nabla_{a_i}Q^{\boldsymbol{\pi}}(\boldsymbol{o},\boldsymbol{a})] \tag{5}$$

$$\nabla_{\theta^c} J(\theta^c) = \mathbb{E}_{\boldsymbol{o}\sim\rho^\pi, \boldsymbol{a}\sim\pi_{\theta_i}}[\nabla_{\theta^c} M(enc\_m_1, ..., enc\_m_N; \theta^c) \cdot \nabla_M \boldsymbol{\pi}_{\theta_i}(o_i, M)\cdot\nabla_{a_i}Q^{\boldsymbol{\pi}}(\boldsymbol{o},\boldsymbol{a})] \tag{6}$$

IV. Experimental Evaluations

In this section, we present experimental results and analyze the performance of our approach compared to several baselines in the edge caching environment, where we investigate the bandwidth efficiency of our approach in a real-world application. The baseline approaches are described below.

No Communication (MADDPG): This is introduced in [15], MADDPG has a central critic network for all agents, but does not have a process of generating and integrating messages. Since our approach is developed from the basic framework of MADDPG, comparing with it is similar to an ablation study of no communication.

Full Sharing: This model directly uses the full observations of each agent as the message shared with every other agent, derived from recent works [7]. This will help us understand the bandwidth efficiency of our method in a comparison study.

Rule-based algorithms: We will also compare with the least recently used (LRU) and least frequently used (LFU) caching policies in the edge caching environment.

CSNet achieves better performance. To test the caching performance of the trained model, we compare CSNet with two rule-based caching strategies, LRU and LFU, apart from the learning-based ones. There are $U = 500$ users, $F = 200$ files, $E = 5$ agents, and the number of neighboring edge servers is $\mathcal{N} = [2, 3, 1, 2, 2]$ due to different distances between them.

We evaluate on a range of cache capacities, with the capacity of each edge's cache ranging from 4% to 15% of F. Fig. 3 show the performance results that are averaged across all agents over 10 runs, each lasts 1000 time steps.

In Fig. 3b, the higher and lower lines of the filled area depict the total hit ratio and the neighbor hit ratio respectively. We observe that as the cache capacity grows, the hit ratio including the neighbor hit ratio increases. This phenomenon can be explained by the fact that edges with a larger cache size

can cache more content within one time step to satisfy more users' content requests. The CSNet achieves a higher total cache hit ratio than both the rule-based algorithms and the scheme without any communication. Though both learning-based algorithms achieve high neighbor cache hit ratios, the one with the presence of efficient communication, *i.e.,* CSNet, makes the most of neighbor cache cooperation.

In terms of other metrics, CSNet still beats the other caching solutions. Fig. 3a shows that CSNet minimizes the transmission latency for all different cache capacities. From Fig. 3c, we observe that CSNet has learned to execute cache replacement catering to users' preferences on content with less cost, compared to approaches with no communication.

Note that the models are only trained in a single scenario with 4% cache capacity. From the results above, we observe that CSNet can help agents adapt to larger cache capacities that are different from what it has experienced during training.

Tradeoff between learning performance and message size. We explored the relationship between the caching performance and the different levels of message size by changing the message compression level. This was realized by tuning the output layer size of message encoder component. Fig. 4 demonstrates that with message compression, CSNet can achieve a higher hit ratio and lower latency, which supports our claim that message encoding and decoding steps can not only help reduce the communication cost but also help knowledge understanding between agents. In particular, the model with a message size of 320 bits obtains the highest cache hit ratio compared to other communication schemes and models with other message sizes, which is $1.33\times$ as much as that of no communication. Meanwhile, the communication overhead of this model is only 2.5% of that resulted by full observations sharing.

Overall, the communication cost is reduced while high performance is maintained. However, a smaller message size (of 160 bits) isn't guaranteed to achieve a higher hit ratio. The optimal message size can differ when the number of agents or the complexity of the network change. To balance the communication overload and learning performance, we are careful about the choice of the message size induced by the message compression component.

V. Concluding Remarks

In this paper, we proposed *CSNet*, a new multi-agent deep reinforcement learning framework incorporating efficient information sharing for wireless edge caching. In *CSNet*, we combine intra-step and inter-step communication by using successive neural networks to adaptively convert local intelligence to restricted-sized messages, selectively aggregate shared messages into a piece of global knowledge, and also retain history messages. In this way, our approach reduced communication overhead while keeping the essential information in messages to maintain good performance. Our learned bandwidth-efficient communication strategies can facilitate agents' caching behavior over both rule-based and learning-based caching algorithms.

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 Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[2] E. Zeydan, E. Bastug, M. Bennis, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, "Big data caching for networking: Moving from cloud to edge," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 36–42, 2016.

[3] W. Jiang, G. Feng, S. Qin, and T. S. P. Yum, "Efficient D2D content caching using multi-agent reinforcement learning," in *Proc. IEEE INFOCOM Workshop*, 2018, pp. 511–516.

[4] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep multi-agent reinforcement learning based cooperative edge caching in wireless networks," in *Proc. IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[5] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, "Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach," in *Proc. IEEE INFOCOM*, 2020, pp. 2499–2508.

[6] H. Zhu, Y. Cao, X. Wei, W. Wang, T. Jiang, and S. Jin, "Caching transient data for internet of things: A deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2074–2083, 2019.

[7] P. Naghizadeh, M. Gorlatova, A. S. Lan, and M. Chiang, "Hurts to be too early: Benefits and drawbacks of communication in multi-agent learning," in *Proc. IEEE INFOCOM*, 2019, pp. 622–630.

[8] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Proc. 30th International Conference on Neural Information Processing Systems (NeurIPS)*, 2016, pp. 2252–2260.

[9] J. Jiang and Z. Lu, "Learning attentional communication for multi-agent cooperation," in *Proc. 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, 2018, pp. 7265–7275.

[10] S. Q. Zhang, Q. Zhang, and J. Lin, "Efficient communication in multi-agent reinforcement learning via variance based control," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 3235–3244.

[11] Z. Ding, T. Huang, and Z. Lu, "Learning individually inferred communication for multi-agent cooperation," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 22 069–22 079, 2020.

[12] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Proc. 30th International Conference on Neural Information Processing Systems (NeurIPS)*, 2016, pp. 2145–2153.

[13] A. Das, T. Gervet, J. Romoff, D. Batra, D. Parikh, M. Rabbat, and J. Pineau, "Tarmac: Targeted multi-agent communication," in *International Conference on Machine Learning*. PMLR, 2019, pp. 1538–1546.

[14] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," in *Proc. AAAI Conference on Artificial Intelligence*, 2018.

[15] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. 31st International Conference on Neural Information Processing Systems (NeurIPS)*, 2017, pp. 6382–6393.

[16] C. Amato, G. Chowdhary, A. Geramifard, N. K. Üre, and M. J. Kochenderfer, "Decentralized control of partially observable markov decision processes," in *Proc. 52nd IEEE Conference on Decision and Control*, 2013, pp. 2398–2405.

[17] 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," in *International Conference on Learning Representations (ICLR) Poster*, 2016.

[18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. The MIT Press, 2018.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[20] H. Mao, Z. Zhang, Z. Xiao, Z. Gong, and Y. Ni, "Learning agent communication under limited bandwidth by message pruning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5142–5149.