Neuro-DCF: Design of Wireless MAC via Multi-Agent Reinforcement Learning Approach

Sangwoo Moon, Sumyeong Ahn, Kyunghwan Son, Jinwoo Park, and Yung Yi
Korea Advanced Institute of Science and Technology
Daejeon, South Korea
{mununum,sumyeongahn,kevinson9473,jinwoo520528,yiyung}@kaist.ac.kr

ABSTRACT

The carrier sense multiple access (CSMA) algorithm has been used in the wireless medium access control (MAC) under standard 802.11 implementation due to its simplicity and generality. An extensive body of research on CSMA has long been made not only in the context of practical protocols, but also in a distributed way of optimal MAC scheduling. However, the current state-of-the-art CSMA (or its extensions) still suffers from poor performance, especially in multi-hop scenarios, and often requires patch-based solutions rather than a universal solution. In this paper, we propose an algorithm which adopts an *experience-driven* approach and train CSMA-based wireless MAC by using deep reinforcement learning. We name our protocol, Neuro-DCF. Two key challenges are: (i) a stable training method for distributed execution and (ii) a unified training method for embracing various interference patterns and configurations. For (i), we adopt a *multi-agent* reinforcement learning framework, and for (ii) we introduce a novel graph neural network (GNN) based training structure. We provide extensive simulation results which demonstrate that our protocol, Neuro-DCF, significantly outperforms 802.11 DCF and O-DCF, a recent theory-based MAC protocol, especially in terms of improving delay performance while preserving optimal utility. We believe our multi-agent reinforcement learning based approach would get broad interest from other learning-based network controllers in different layers that require distributed operation.

CCS CONCEPTS

• **Networks** → **Link-layer protocols**; • **Computing methodologies** → **Multi-agent reinforcement learning**.

KEYWORDS

Wireless MAC, Optimal CSMA, Multi-agent RL

ACM Reference Format:
Sangwoo Moon, Sumyeong Ahn, Kyunghwan Son, Jinwoo Park, and Yung Yi. 2021. Neuro-DCF: Design of Wireless MAC via Multi-Agent Reinforcement Learning Approach. In *The Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc '21), July 26–29, 2021, Shanghai, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3466772.3467043

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MobiHoc '21, July 26–29, 2021, Shanghai, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8558-9/21/07…$15.00
https://doi.org/10.1145/3466772.3467043

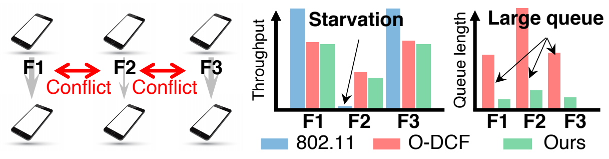


Figure 1: FIM topology example.

1 INTRODUCTION

802.11 DCF (Distributed Coordination Function) has often been reported to suffer from coordinating both throughput and fairness. A representative example, depicted in Figure 1, is when three flows, say $F1$, $F2$, and $F3$, form a so-called flow-in-the-middle (FIM) topology, where each of the two flows $F1$ and $F3$ conflict with the flow $F2$, while $F1$ and $F3$ do not conflict with each other. In this case, 802.11 DCF experiences a serious starvation of the "middle" flow $F2$, whose performance becomes almost zero. Over the last two decades, there has been an extensive array of research [12, 34, 41] to solve these problems of 802.11 DCF.

Recent interesting approaches, e.g., O-DCF [19] and A-DCF [18], redesigned the way of controlling the parameters of CSMA (Carrier Sense Multiple Access), by adopting some theoretical results (often called optimal CSMA [15]). These approaches utilize additional local information (e.g., queue length) and respond to network behaviors (e.g., collisions) in a differentiated manner, so as to implicitly understand the neighboring interference patterns. Despite some degree of improvement of O-DCF and A-DCF, relying on queue length only is not enough to fully incorporate various interference patterns. Thus, even in O-DCF and A-DCF, there are non-negligible "patched-up" engineering solutions which often have poor performances in some metric such as delay [17] (see Figure 1) and compatibility with TCP [18]. They are also unprincipled solutions to this problem and thus, they may have sub-optimal performances in cases which they have not been exhaustively tested.

In this paper, we take a learning-based approach to train a practical wireless MAC controller, where we call the resulting MAC protocol *Neuro-DCF*. Our protocol is inspired by the recent breakthrough of Deep Reinforcement Learning (DRL) [25], which introduces an *experience-driven* or *data-driven* approach for training the controller from experience samples without explicitly knowing the model structure. This learning-based approach is showcased to be capable of solving complicated games with extremely large state space. DRL has already received attention from other fields in the networking community, and several latest proposals have come out to replace traditional rule-based networking protocols such as traffic engineering or congestion control [14, 28, 43]. The goal of this paper is to develop an experience-driven MAC and qualitatively and quantatively investigate the potential of experience-driven MAC.

There are two key design requirements that must be satisfied when developing a practical wireless MAC using an experience-based approach. We henceforth summarize them below in conjunction with the challenges and our design choices.

1) ***Training for distributed operation:*** Coordinating the wireless nodes for the medium access should involve minimal or entirely no communication, as in 802.11 DCF. One simple approach is to assume a single coordinator in training which is aware of all network status and is in charge of operating all wireless nodes. However, this does not allow distributed operation of nodes in execution. To tackle this, we formulate the problem as a cooperative multi-agent RL (MARL) problem, where all RL agents are trained to optimize a common objective. One naïve approach is to train every agent in the environment separately, treating other agents' behaviors as part of the environment. This approach does not work, especially in complex scenarios, because of the non-stationarity problem caused by the change in policies of other agents. Thus, it is necessary to understand the complex interplay among multiple controllable entities and to develop a training structure which induces cooperation even without exchanging status information among agents.

   Our choice: We adopt a recently introduced paradigm called Centralized Training with Decentralized Execution (CTDE). CTDE has a centralized training scheme which effectively utilizes the joint experiences to learn decentralized policies for the agents which can be executed separately. This paradigm has been used in many MARL approaches, e.g., [7, 23], but the key design choice lies in which RL algorithm to choose and how to incorporate the policy evaluation and improvement modules regarding distributed multi-agent operations, all of which depends on the target task. We use Proximal Policy Optimization (PPO) algorithm [33] as our baseline RL algorithm, which is known to be both sample efficient and stable at training, and we extend it for multi-agent scenarios. See Figure 2 for the overall training structure of our Neuro-DCF.

2) ***Avoiding per-scenario training.*** A classical RL training in the MAC scheduling task is to have separated trainings for every specific network topology and the number of nodes/links. This is extremely challenging to train in MARL, unless the state space and the number of agents is relatively small. It would be highly beneficial if our MAC controller is applicable to real-world networking environments. To make our MAC controller practical, the trained controller should incorporate as many general interference topologies and network scales as possible. Our objective is to generalize a network controller to multiple interference structures without retraining on every single instances, and to come up with a policy that is capable of dealing with a wide range of interference scenarios.

   Our choice: In order to solve the generalization objective, we make three-fold design choices. First, we adopt a *graph neural network (GNN)* [16] structure to represent the interference relationships among links. Such GNN representations are used for rendering the generalized value function, which is an essential component of reinforcement learning. A GNN takes graph structured inputs and outputs an embedded feature representations. The common approach is to design a *graph convolutional* layer which operates similarly to the convolutional neural network (CNN). The graph convolution works by applying a filter which summarizes the features of a given node and its neighboring nodes. Utilizing the graph convolution structure, GNN is suitable for the parameterized representation of *the local interactions* between agents, and the number of parameters does not increase with respect to the size of the input graph.
   Second, in order to effectively train our GNN value function, we introduce a training method called *random graph training*, which serves a similar purpose as mutli-task RL [39]. In this method, we choose randomly sampled interference graphs from a given distribution. We generate different graphs for each episode, and train the wireless MAC controller under different environments using the same value network. The random graph training ensures our policy to behave efficiently in the graphs within the distribution used in the training phase.
   Finally, we suggest to use a *parameter shared* (PS) structure, where all controllers have a same set of model parameters. This design complies to the general design principle of the network protocol, where the nodes in the network have to operate according to the same algorithm. The PS architecture has two benefits in our framework. First, the number of parameters does not change as the scale of network grows, thereby enabling our algorithm to be easily adapted to large scale networks. Second, without PS, it is nontrivial to deploy the trained agents onto the network since we have to additionally decide which model to deploy on each wireless nodes. With the aid of GNN and PS, we ensure that the mentioned requirements are satisfied.

We provide extensive evaluation results using the ns-3 network simulator [31], from simple interference topologies such as fully-connected (FC) to large, complex and general structures. We compare our trained MAC algorithms with 802.11 and O-DCF in terms of significant performance measures like fairness and delay. The results indicate that Neuro-DCF outperforms 802.11 in terms of network utility and achieves superior delay performance compared to O-DCF without any explicit control-plane communication during execution. In particular, we check that our Neuro-DCF achieves up to 204.4% gain in total utility over 802.11 and queue length decreases by up to 83.5% over O-DCF. Moreover, additional studies regarding the generalization ability of Neuro-DCF show that our random graph training covers a reasonable area of generalization over graph distribution.

Related work. The performance problem of 802.11 DCF has been repeatedly reported and tackled by numerous papers. The initial effort proposed the idea of dynamically adjusting the contention window (CW) under 802.11 DCF [3, 12], and the following papers came out to study about practical implementation issues [9, 34]. However, these early solutions are limited to specific topologies such as fully connected (FC) and did not guarantee any improvements on more problematic topologies such as FIM. To achieve the optimal wireless MAC in general topology, the theoretical insights were given by the optimal CSMA [15], with the specific adjusting rules of CSMA parameters associated with the queue length. However, the optimal CSMA algorithms are generally not considered as a practical algorithm since they make some unrealistic assumptions for theoretical soundness (See [44] for details). Practical versions like O-DCF [19] or A-DCF [18] have shown that the optimal CSMA is reasonably effective by introducing some case-by-case engineering solutions. These kinds of heuristic approaches are not guaranteed to work properly in generalized situations and might be prone to unpredictable failure scenarios.

There has been a series of studies regarding the experience-driven approach on access control. Early works assumed a simplified finite-state Markov channel (FSMC) to apply simple RL algorithms such as Q-learning, which is targeted for efficiency or delay [24]. Since DRL was introduced, it has now become possible to learn complex channel dynamics with deep neural networks. Therefore, the access control problem has been re-investigated using these DRL techniques. Single-agent DRL is applied to the problem by assuming a single controlling entity, such as cognitive radio [40] or scheduling in cellular networks [5] by base station control.

Since the access control is a multi-agent problem in nature, there have been some works applying DRL in the multi-agent setting. These works are largely divided into *cooperative* and *competitive* problems. Examples of competitive setting include unlicensed spectrum management in LTE [4] or independent ALOHA [20]. The cooperative setting considers the problem as a global optimization problem with a single objective. [28] is the most closely related work to ours as it targets multiple spectrum access problem for optimizing proportional fairness (PF) objective by formulating the problem as a cooperative MARL problem.

DRL has been applied to other network control domains, such as traffic engineering [43] and congestion control [14]. [43] modified the DDPG [22] algorithm for the traffic engineering problem which involves bandwidth management of multiple co-existing flows. [14] provided DRL-based congestion control in TCP and showed significantly improved performance compared to the current CUBIC congestion control.

2 BACKGROUND

2.1 Deep Reinforcement Learning (DRL)

Reinforcement Learning (RL) [35] has been widely used in various decision making problems because of its ability to evaluate and optimize the expected sum of desired rewards, without relying on prior knowledge of the problem. RL uses Markov Decision Process (MDP) as a mathematical formulation.

A fully-observable MDP is defined as a tuple $\mathcal{G} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$. At each discrete time step $t \in \{0, 1, \cdots, \}$, an RL *agent* observes the *state* $s_t \in \mathcal{S}$, and selects an *action* $a_t \in \mathcal{A}$, according to the stochastic mapping rule called *policy* $\pi(a|s) : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$. Given the action from the agent, the environment changes its state depending on the *transition* $P(s'|s, a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$. After the transition, the *reward* r_t is decided from the reward function $r(s, a) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, and the state in a next step s_{t+1} is given to the agent. The objective of RL problem is to obtain an optimal policy π^* that maximizes the expected discounted sum of reward $\mathbb{E}_{s_t, a_t \sim \pi}[\sum_t \gamma^t \cdot r_t]$. The *discount factor* $\gamma \in [0, 1)$ is often used to represent the shrinked horizon of the problem for mathematical tractability. The commonly used technique is called *function approximation*, in which we parameterize the policy π and value function Q as π_θ and Q_w, respectively. With the power of deep neural networks, we can represent the policy and value function with a higher level of abstraction. [25].

One way to solve the RL problem is to iteratively apply the gradient to optimize π_θ using the formula called *policy gradient* [36], written as

$$L^{\text{PG}}(\theta) = \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[\log \pi_\theta(a_t|s_t) G_t \right], \tag{1}$$

where $L^{\text{PG}}(\theta)$ is the objective function. The future cumulative reward $G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ can be estimated with the advantage function A_w without introducing any bias, and the policy gradient becomes

$$L^{\text{PG}}(\theta) = \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[\log \pi_\theta(a_t|s_t) A_w(s_t, a_t) \right]. \tag{2}$$

The advantage A_w is often estimated as a temporal difference (TD) residual given by

$$A_w(s_t, a_t) \approx \mathbb{E} \left[r_t + \gamma V_w(s_{t+1}) - V_w(s_t) \right], \tag{3}$$

given the state-value function $V_w(s_t)$. Using the value function to estimate the policy gradient is specifically called *actor-critic*, as the state-value function V_w (called *critic*) is used to derive the update equation of the policy π_θ (called *actor*).

2.2 Multi-agent RL

In multi-agent RL, each agent has only partial information about the entire state over time, based on which it chooses its action. This partial information is often a local information, which naturally leads to distributed execution. Due to this reason, the following decentralized partially observable MDP (Dec-POMDP) [29] is used as the de facto standard for modeling MARL, i.e., a tuple $\mathcal{DG} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma, \mathcal{N}, O, Z \rangle$, where $s \in \mathcal{S}$ denotes the true state of the environment. Each agent $i \in \mathcal{N} := \{1, \cdots, N\}$ chooses an action $a_t^i \in \mathcal{A}$ at each time step t, [1] giving rise to a joint action vector, $\mathbf{a} := [a^i]_{i=1}^N \in \mathcal{A}^N$. Function $P(s'|s, \mathbf{a}) : \mathcal{S} \times \mathcal{A}^N \times \mathcal{S} \mapsto [0, 1]$ governs all state transition dynamics. Here, we focus on the cooperative case, namely every agent shares the same joint reward function $r(s, \mathbf{a}) : \mathcal{S} \times \mathcal{A}^N \mapsto \mathbb{R}$ [2], and $\gamma \in [0, 1)$ is the discount factor. Each agent has its individual, partial observation $o^i \in O$ according to some observation function $Z(s, i) : \mathcal{S} \times \mathcal{N} \mapsto O$. Each agent also has an action-observation history $\tau^i \in \mathcal{T} := (O \times \mathcal{A})^*$, on which it conditions its stochastic policy $\pi^i(a^i|\tau^i) : \mathcal{T} \times \mathcal{A} \mapsto [0, 1]$. Each agent conditions its policy on the history of observations if the state is partially observable.

Why hard? The Dec-POMDP problem is known to be challenging because it is not straightforward to apply existing single-agent RL algorithms to solve it. The key difficulty comes from the *non-stationarity* phenomenon [11]. With a single-agent RL perspective, each agent would formulate a single-agent value function, denoted

[1] We denote the agent index in a superscript and the time index in a subscript of notations.

[2] If the agent i has its own interest of the reward function $r^i(s, \mathbf{a})$, we call this formulation a Partially Observable Stochastic Game (POSG), which is a more general form of (not necessarily cooperative) MARL.

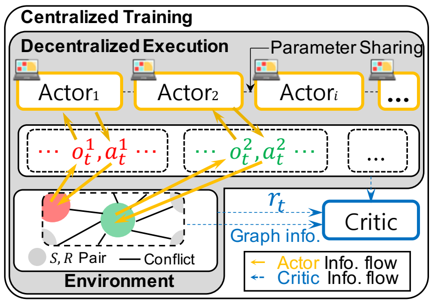


Figure 2: The depiction of the centralized training with decentralized execution (CTDE) architecture in wireless MAC.

as $Q^i(\tau^i, a^i)$. This perspective treats the other agents as part of the environment which the agent has to interact with. However, the policy of each of the other agents, denoted as π^{-i}, is changed throughout the training, which makes the perceived environment to change. The MDP from the single agent perspective is no longer stationary at this point, and the training stability is not guaranteed. Indeed, naïvely applying single-agent algorithms demonstrated underwhelming results [37]. Therefore, research on cooperative MARL [7, 23] is focused on proper evaluation and improvement of the joint policy.

3 DESIGN OF NEURO-DCF

We next provide the explanation of our approach which solves the wireless MAC problem with reinforcement learning. We first explain the overall structure of the centralized training with decentralized execution. Then, we discuss the problem formulation and the definition of our MDP. Next, we describe our algorithmic approaches to solve the problem setting.

3.1 Overall Training Structure

Recently, there have been advances in techniques to train decentralized policies, where we can fully access the simulation or laboratory environment during the training process. This *Centralized Training with Decentralized Executions* (CTDE) principle became popular in multi-agent planning, as it can provide a general framework for cooperative yet decentralized MARL. The actor-critic architecture particularly fits well with the CTDE framework as we can separate the role of the actor and critic, and differentiate the information flow onto these components. The actor is responsible for the agent's action selection using its local observations, and the critic comprehends the overall state and action information and predicts the future rewards. The reward to be predicted is a *global* reward, which is the natural choice when devising a cooperative objective. Figure 2 describes the different information flows of actor and critic. Some actor-critic based CTDE algorithms involving deep neural networks have came out lately, following the abovementioned design principle [7, 23]. In these works, the actor usually takes a form of recurrent neural networks to encode the trajectory τ^i, and operates in a decentralized manner. The critic estimates the value of joint actions by representing the Q-function as $Q(s, \mathbf{a})$. This *centralized* representation of the critic can evaluate cooperative actions and enforce the actors to coordinate.

Figure 2 represents the CTDE architecture of our framework. According to the figure, the actor collects its local observation o_t^i to decide its action a_t^i, and the critic receives the overall joint observation $[o_t^i]_{i=1}^N$ and action $[a_t^i]_{i=1}^N$, along with the graph structure of the environment. Here, we assume that the true state s_t can be approximated with the joint observation of the agents $[o_t^i]_{i=1}^N$. For the training environment, we configure the ad-hoc network with multiple one-hop UDP flows. It is assumed that all sender-receiver pairs are disjoint, and the interference relationships are represented as an undirected graph (see Figure 5a for example). The node in the interference graph represents the one-hop flow, and the edge represents the interference relationship between two flows. Along with the joint observation and joint action, our critic also evaluates this graph structure to effectively understand the environment. The critic predicts the value function and is trained with the cooperative reward signal r_t. The critic estimation is used to derive the gradient for the policy update by (2). Note that the critic execution and reward signaling are only used in the training phase, and we require only the trained actor model in execution.

3.2 Observations, Actions, and Rewards

We now define the *observation*, *action*, and *reward* functions, respectively. The *agent* is the wireless sender in our problem, and it is in charge of gathering the local observations and executing the actions accordingly. A single cooperative reward is given to the agents only at training time. Once the trained policy is deployed, the nodes can operate in a completely decentralized manner. We explain the design rationale and details of our selection for the observations, actions, and reward in the following paragraphs.

Observation. Since we model the problem as a Dec-POMDP task, each observational feature must be locally perceptible to the agents. Additionally, the chosen observations are completely accessible using the standard 802.11 radio equipment without using additional sensory hardwares, so that it can be applied just by modifying the software layer. The observational features are defined as represented in Table 1. We denote the observation vector of agent i at time t by o_t^i. The observation includes the measurements of the network statistics, i.e., link throughput and the end-to-end delay. The network statistics are collected for a fixed monitoring interval (MI). Since 802.11 MAC layer provides the Acknowledgement (ACK) frame to indicate the success of the transmission, we utilize the ACK frame to estimate transmission rate, delay and frame error rate. The average frame delay is calculated as an exponential weighted moving average (EWMA) estimate of per-frame end-to-end delay measured from the ACK frame timestamp. In addition, the MAC layer internal states, such as the backlog length and current transmission parameters, are also included in o_t^i. The indirect neighbor detection is possible by the Clear Channel Assessment (CCA) mechanism, which is an essential component of CSMA.

Action. Figure 3 describes our RL agent's MAC layer control structure. In order to minimize the frame delay, we control both arrival and service processes of the MAC layer by jointly control the traffic input and MAC. Accordingly, we have a two-dimensional action space, and we make use of two adjoint controllers in execution. The rate control action $a_{t,\mathrm{RC}}^i$ takes the role of adjusting enqueue rate from Control Queue (CQ) to Medium Access Queue (MAQ), as

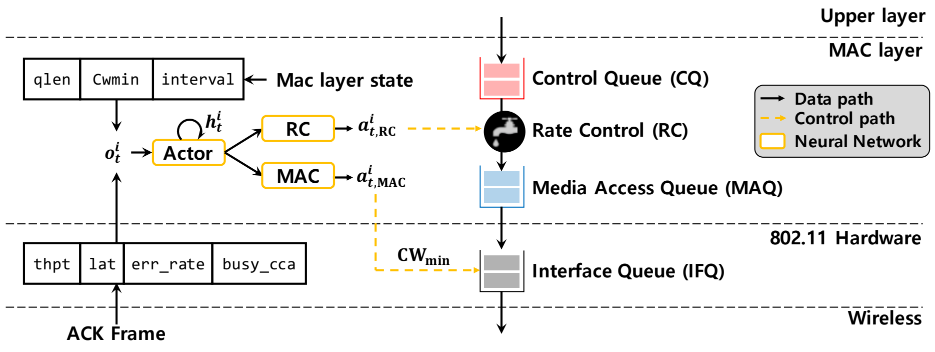


Figure 3: The overall operational structure of the RL actor to the MAC layer. The observational features can be collected locally without any communications. We use LSTM neural network with hidden state h_t^i and sequential observations o_t^i to encode the trajectory τ_t^i.

Feature	Description
thpt	The instantaneous throughput
qlen	The length of MAQ
lat	EWMA estimate of the frame delay
err_rate	Frame error rate
busy_cca	The fraction of time when the CCA state is either RX or CCA_BUSY
CW	Current selection of CW
interval	Current selection of frame injection interval

Table 1: The observation features used by an agent

shown in Figure 3. The rate control action is chosen between evenly spaced discrete numbers as unit of packets per second (pps). The MAC action $a^i_{t,\mathrm{MAC}}$ decides the Contention Window (CW), which dictates the backoff period before channel access in the CSMA algorithm. The CW parameter is chosen as $2^n - 1$, where $n \in \{0, \cdots, 8\}$. The backoff timer is randomly chosen from the interval $[0, \mathtt{CW}]$, therefore preserving some randomness for collision avoidance.

Reward. Now we propose our reward function which reflects the queuing delay performance and the utility performance. We are incorporating a cooperative Dec-POMDP setting, where the reward function is shared among all agents. We define our reward function with two components as

$$r_t := r_{t,\mathrm{NUM}} + \beta r_{t,\mathrm{queue}}, \tag{4}$$

where β is the balancing parameter between the components. Note that the reward function could be designed differently to cope with other design objective than utility or delay, and it would be an interesting future research to consider multiple other rewards with our CTDE design.

The first component $r_{t,\mathrm{NUM}}$ reflects the network utility maximization (NUM) objective, which is calculated by measuring the average throughput of each agent. We call this reward component as *utility reward.* Thus, we define our reward functions as

$$r_{t,\mathrm{NUM}} := \sum_i U(\tilde{x}_t^i). \tag{5}$$

where $U(\cdot)$ is the *utility function.* The utility function is a continuous, increasing, and strictly concave function to reflect the *diminishing return* of the rate. When we specifically choose the $U(x) = \log(x)$, the objective represents proportional fairness (PF), and it is recognized to be the unified measure of efficiency and fairness. $\tilde{x}^i$ indicates the EWMA-approximated throughput of the link i, calculated as

$$\tilde{x}_t^i := (1-\alpha)\tilde{x}_{t-1}^i + \alpha x_t^i, \tag{6}$$

where x_t^i is the instantaneous throughput of the agent i at time step t, and α is a smoothing constant. This moving average NUM reward expresses the short-term estimate of the long-term utility. During training, the algorithm is reinforced to maximize the given reward, which results in maximizing the short-term utility in every time step, eventually contributing to the long-term utility maximization.

The second component $r_{t,\mathrm{queue}}$ represents the queueing delay objective. Following Little's theorem, the average queuing delay is equivalent to the average queue length. Therefore, we define our delay component as a negative sum of queue length of the agents,

$$r_{t,\mathrm{queue}} := -\sum_i q_t^i, \tag{7}$$

and we call this reward component as *queue reward.* Here q_t^i denotes the MAQ length of the agent i at time step t.

We now explain the details of our Neuro-DCF algorithm to solve the Dec-POMDP environment defined previously. The wireless MAC problem has several requirements and challenges when viewed from RL perspective, such as non-stationarity. The overall structure of Neuro-DCF is outlined in Figure 2, where our algorithm is divided into actor and critic components. Figure 3 and 4 depicts the actor and critic structure, respectively. In Figure 3, we encode the history τ_t^i by using an LSTM network with input o_t^i and hidden state h_t^i. In Figure 4, every Dense and Conv blocks represent a neural network with a set of trainable parameters. We further demonstrate detailed description of our algorithm. We first present an extended version of PPO [33] for updating the actors in the cooperative MARL problem. Next, we propose a novel critic architecture with graph embedding to apply to arbitrary network topologies. Finally, we introduce parameter sharing that learns a single shared policy for multiple actors simultaneously.

3.3 Multi-Agent Proximal Policy Optimization (MAPPO)

Primer of PPO. We take Proximal Policy Optimization (PPO) [33] as our baseline RL algorithm, which is known to be one of the most efficient and stable policy gradient algorithms up to date. The

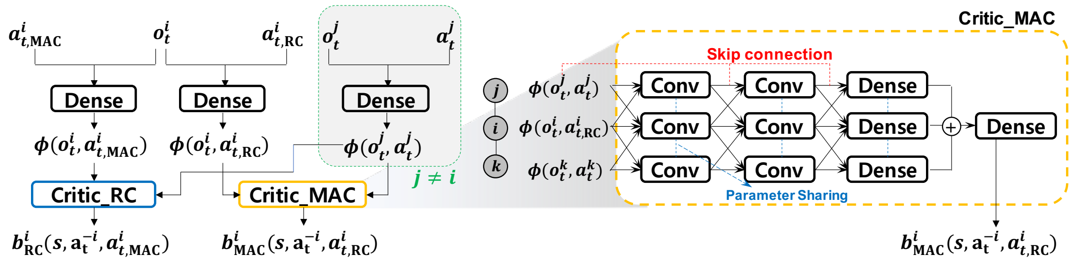

$a^i_{t,\mathrm{MAC}}$
o^i_t
$a^i_{t,\mathrm{RC}}$
o^j_t
a^j_t
Dense
Dense
Dense
$\phi(o^i_t, a^i_{t,\mathrm{MAC}})$
$\phi(o^i_t, a^i_{t,\mathrm{RC}})$
$\phi(o^j_t, a^j_t)$
$j \neq i$
Critic_RC
Critic_MAC
$b^i_{\mathrm{RC}}(s, \mathrm{a}^{-i}_t, a^i_{t,\mathrm{MAC}})$
$b^i_{\mathrm{MAC}}(s, \mathrm{a}^{-i}_t, a^i_{t,\mathrm{RC}})$
Critic_MAC
Skip connection
j
i
k
$\phi(o^j_t, a^j_t)$
$\phi(o^i_t, a^i_{t,\mathrm{RC}})$
$\phi(o^k_t, a^k_t)$
Conv
Conv
Dense
Dense
Parameter Sharing
$b^i_{\mathrm{MAC}}(s, \mathrm{a}^{-i}_t, a^i_{t,\mathrm{RC}})$

Figure 4: The centralized critic architecture of our algorithm.

PPO algorithm utilizes the same experience trajectory multiple times, by enforcing the policy update constraint while we apply the policy gradient. PPO restricts the policy update ratio within the *trust region* by clipping the advantage functions, i.e. the objective function becomes:

$$L^{\mathrm{PPO}}(\theta) = \mathbb{E}_{s_t, a_t \sim \pi_\theta}\left[\mathrm{ppo\_clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}, \epsilon, \hat{A}_t\right)\right]. \tag{8}$$

Here $\mathrm{ppo\_clip}(\cdot, \epsilon, \hat{A}_t)$ denotes the *clipping* function specifically defined in [33] to prevent excessively large policy updates, θ_{old} are the parameters before the update, and $\hat{A}_t$ is the advantage function calculated by Generalized Advantage Estimation (GAE) [32], as

$$\begin{aligned} \hat{A}_t &= \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t-1}\delta_{T-1}, \\ \text{where} \quad \delta_t &= r_t + \gamma V_w(s_{t+1}) - V_w(s_t). \end{aligned} \tag{9}$$

In the above equation, λ is a GAE hyperparameter. The GAE advantage generalizes the one-step TD residual in (3).

MAPPO. However, since PPO is a single-agent RL algorithm, it is non-trivial to apply PPO to solve a multi-agent problem. We propose *multi-agent PPO (MAPPO)* algorithm as the multi-agent extension of PPO. In MAPPO, the multi-agent advantage of agent i is given as

$$\begin{aligned} \hat{A}^i_t &:= \delta^i_t + (\gamma\lambda)\delta^i_{t+1} + \cdots + (\gamma\lambda)^{T-t-1}\delta^i_{T-1}, \\ \text{where} \quad \delta^i_t &:= r_t + \gamma b^i_w(s_{t+1}, \mathrm{a}^{-i}_{t+1}) - b^i_w(s_t, \mathrm{a}^{-i}_t). \end{aligned} \tag{10}$$

Here, we substitute the value function $V_w(s_t)$ into $b^i_w(s_t, \mathrm{a}^{-i}_t)$, which is defined as the *counterfactual baseline.* [3] $b^i_w(s_t, \mathrm{a}^{-i}_t)$ estimates the future reward of the joint action *excluding* the action of agent i. The term "counterfactual" comes from the deductive reasoning process to determine the impact of agent i's action to the value. This idea first came out in [7], but we largely enhanced [7] by combining it with a trust region idea, thus improving the sample efficiency.

Since we have to jointly decide the rate control (RC) and MAC actions, we decompose our actor architecture to separately execute each action, denoted by $a^i_t = (a^i_{t,\mathrm{RC}}, a^i_{t,\mathrm{MAC}})$. For training multiple actions, we adopt the approach from [42] to calculate each action's advantage separately. The multi-agent counterfactual baseline $b^i(s, \mathrm{a}^{-i})$ is further divided into *per-action counterfactual baselines* as $b^i_{\mathrm{RC}}(s, \mathrm{a}^{-i}, a^i_{\mathrm{MAC}})$ and $b^i_{\mathrm{MAC}}(s, \mathrm{a}^{-i}, a^i_{\mathrm{RC}})$. Consequently, we can further assign the credit to each action components. By considering multi-agent and per-action counterfactuals, the baseline reduces the variance of the gradient without changing expectation. The computations take place in the centralized critic unit.

[3]The superscript $-i$ represents the agent indices excluding agent i.

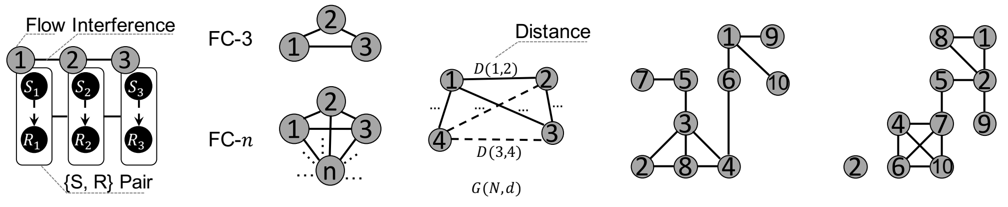
3.4 Graph Embedding Critic

One important aspect of the networking protocol is that the algorithm should be capable of adapting to arbitrary situations. For example, TCP congestion control should be able to run on any given link capacities without changing its operational parameters. This is also the case in our MAC controller design, where the agent has to function in arbitrary network structures. This requirement becomes especially challenging when we use learning-based methods as it is trained using samples of interplay between agents. Even though we train our MAC algorithm in one specific environment, the algorithm has no guarantee to work properly in other environments.

Graph neural networks. To overcome the challenge, we need a consistent representation of the critic regardless of the changes in graph topology. We design our multi-agent critic with a *graph neural network (GNN)* structure. In the last few years, GNNs have shown its great usefulness on graph-structured domains such as social networks [16] or protein-interaction networks [6]. The most widely used method is *graph convolutional network (GCN)* [16], which updates the node representations from one layer to the other, with similar operation as the convolutional neural network. The convolutional layer integrates the feature vectors from neighbor nodes and generates the latent feature vector. Note that the update only depends on the neighborhoods, independent of graph size. The graph-structured connectivity is also applied to represent the multi-agent interaction, and exploiting the graph structure in a cooperative MARL has proven to be effective in certain areas [1, 27].

Figure 4 shows the detailed architecture of our critic. The left side of the figure describes the separated counterfactual calculation of two different control components, i.e., rate control and MAC. Each agent i has two distinct node representations from $\{o^i_t, a^i_{t,\mathrm{MAC}}\}$ and $\{o^i_t, a^i_{t,\mathrm{RC}}\}$, each goes into $\phi(o^i_t, a^i_{\mathrm{MAC}})$ and $\phi(o^i_t, a^i_{\mathrm{RC}})$, respectively. The "Dense" block denotes the fully-connected neural network, and $\phi(\cdot)$ denotes the neural embedding of the inputs, with possibly different parameters. We call these embeddings as *per-action embeddings.* For all $j \neq i$, we calculate the embeddings from $\{o^j_t, a^j_t\}$ and produce $\phi(o^j_t, a^j_t)$, which we call *other-agent embeddings.* All embeddings $\phi(\cdot)$ correspond to the *node features* which are fed into the GNN. We now combine the per-action embeddings and other-agent embeddings to represent the per-action baselines, $b^i_{\mathrm{RC}}(\cdot)$ and $b^i_{\mathrm{MAC}}(\cdot)$. For the baseline calculation, the embedding combination $\{\phi(o^i_t, a^i_{\mathrm{MAC}}), [\phi(o^j_t, a^j_t)]_{j\neq i}\}$ is used to approximate $\{s, \mathrm{a}^{-i}, a^i_{\mathrm{MAC}}\}$, which is used for calculating $b^i_{\mathrm{RC}}(s, \mathrm{a}^{-i}, a^i_{\mathrm{MAC}})$. Calculation of $b^i_{\mathrm{MAC}}(s, \mathrm{a}^{-i}, a^i_{\mathrm{RC}})$ works in a similar manner.

The right side of the figure shows a closer look of the critics *Critic_RC* and *Critic_MAC*, which have the same structure but have



(a) FIM (b) FC-n (c) $G(N,d)$ (d) T1 (e) T2

Figure 5: (a) Flow-in-the-middle (FIM), (b) Fully-connected (FC), (c) Random geometric graph, and (d),(e) Samples of $G(10, 0.3)$. Each node and edge represents {Sender (S), Receiver (R)} pair and interference, respectively.

different parameters. The "Conv" block represents the graph convolutional operation, with shared parameters same as the original GCN paper [16]. The skip connection is adopted from the DenseNet [13] structure, as it is a common practice for preserving the node features in graph neural networks.

Random graph training. To train the GNN critic, we characterize the wireless MAC environment as a *link interference graph* with a set of L links, represented by the adjacency matrix $A \in \{0,1\}^{L \times L}$. An example of the link interference graph is presented in Figure 5. The dashed line represents the interference relationships between one-hop ad-hoc links (A, B, and C in the figure). We aim to generalize our MAC algorithm in a *distribution* of graphs $\mathcal{P}$. For the generalization, the training proceeds in the following steps. When every training episode e starts, the random graph $A^e \sim \mathcal{P}$ is randomly generated from any possible graph distribution $\mathcal{P}$ and each agent executes its current policy to collect the sample experiences. Then, we run a single training iteration out of those samples. With sufficient amount of random graph generation, we can expect the trained policy would be functional on the interference graphs in $\mathcal{P}$.

This method is particularly beneficial because it provides a general design principle to train a protocol that works on a range of different environments. With the aid of smart parameterized representation of environments, we don't have to re-train the model every time when the environment changes. Considering that the RL training process can be sometimes time-consuming, which can take days or even weeks, this generalization technique enables us to make the RL methodology to be practically applicable by training a pseudo-universal policy that covers the spectrum of the environment configurations. Moreover, our approach is not bound to a specific NN architecture as long as the critic can provide the generalized representation over a set of environments. For example, one can use a PointNet [30] architecture to embed a spatially distributed wireless nodes instead of GNN.

3.5 Parameter-Shared Agents

The training parameters we use for the actors are shared across the agents. This structure is selected to keep the number of parameters in control even if we increase the number of agents significantly. As the result, our algorithm is universally applicable to any agent regardless of the agent index, making it a practical network protocol. Our GNN architecture also preserves the scale of parameter size regardless of the number of agents. By sharing the parameters, we can reduce the order of the model size from $O(N)$ to $O(1)$, where N is the number of agents to train. By sharing the parameters [10, 38], we do not have to consider the mapping relationship when we apply our actors in different network environments. The parameter-sharing operation is implemented as a centralized training process, and those controllers can be commonly deployed to the wireless nodes for decentralized operations. Even if the parameters are shared, different actions are taken because the observation history of each agent is different. With the parameter sharing architecture, there could be slight disadvantages when we try to solve in heterogeneous environment, e.g., HetNets or large diverse networking. However, here we are interested in a local ad-hoc wireless network, where we can reasonably assume that the underlying channel conditions are homogeneous.

4 EVALUATION

4.1 Implementation

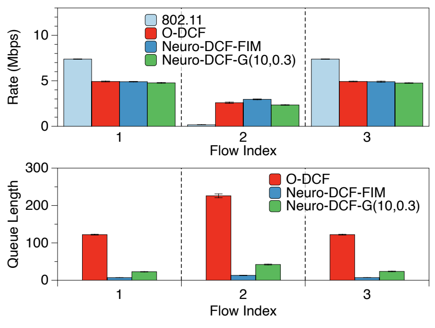
We implemented the wireless MAC environment using the ns-3 simulator [31]. We use ns3-gym framework [8] to connect the C++-based simulator and Python-based OpenAI Gym [2]. As depicted in Figure 3, we make the same two-level queueing architecture as in O-DCF [19]. This choice was made in order to assess a fair end-to-end delay comparison, since the end-to-end frame delay is heavily governed by the queueing structure of the networking stack. We use and modify the PPO algorithm in RLlib [21] and implemented a GNN-based multi-agent PPO algorithm for our use. Our source code is uploaded in `https://github.com/munanum/ns3-gym-csmarl`, along with O-DCF implementations for testing.

4.2 Experimental Setup

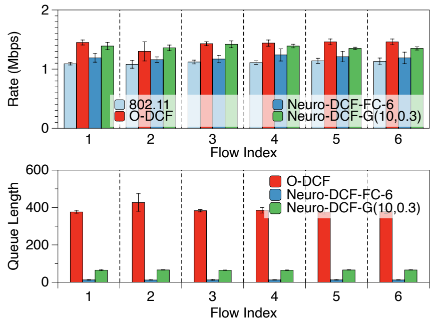
Network configuration. We tested our MAC algorithm under fixed 12 Mbps UDP traffic with 1500 B packets, along with 12 Mbps channel for saturated traffic environment. The RTS/CTS mechanism is disabled since there is no hidden terminal problem in our wireless setting. All observational features are given to the sender by observing the ACK frame and CCA indicators, and no other communications are involved. We assume there is no path loss or fading on wireless channels, and all packets are perfectly delivered when no MAC frame collision occurs. We use a matrix propagation loss model in ns-3, in which there is 0-dB loss in connected flows and ∞-dB loss in disconnected flows. We simulate our network for 20 seconds on each episode, and the monitoring interval is set to 5 ms.

Interference topology. We mainly focus on the three kinds of interference topologies as follows:

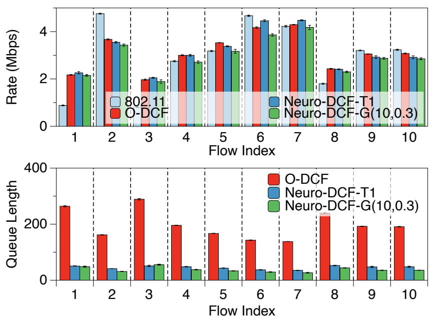
- **Flow-in-the-middle (FIM):** Three flows with two outer flows and one inner flow. The outer flows are only interfering with one flow, but the inner flow interferes with two flows,



(a) FIM



(b) FC-6



(c) T1

Figure 6: First and second row represent average rate and queue length with each topology: (a) FIM, (b) FC-6, and (c) T1.

hence making an *asymmetric* interference relationship which makes a classical example where 802.11 algorithm suffers from starvation.

- **Fully-connected with size n (FC-n):** All n flows are interfering with each other (Figure 5b). The FC topology has a *symmetric* interference structure among flows.
- **Random geometric graph with N flows and d threshold ($G(N, d)$):** N flows are uniform randomly placed in 2-D unit square, and only the flows within the threshold d distance are connected (See Figure 5c for description). Figure 5d and 5e show example realizations of $G(10, 0.3)$. The positional representation of the graph is omitted for brevity.

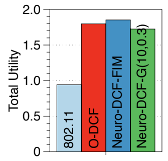
FIM and FC topologies represent *synthetic* topologies, which are simple toy environments yet entail important characteristics of wireless channels. The random geometric graph emulates the spatial placement of interfering wireless nodes, which makes more *realistic* interference scenarios.

Training setup. In order to clarify our Neuro-DCF variants in terms of training topology, we utilize following names which are; (i) Neuro-DCF-$\mathcal{T}$: Trained on single topology $\mathcal{T}$ only, e.g., we call Neuro-DCF-FIM a model from FIM topology. (ii) Neuro-DCF-$G(N, d)$: Trained on various random geometric topologies which are randomly sampled from $G(N, d)$. Note that different variants are separate policy models trained on each topology. We interchangeably call Neuro-DCF-$\mathcal{T}$ as *Neuro-DCF-single* and Neuro-DCF-$G(N, d)$ as *Neuro-DCF-general*, since Neuro-DCF-single is solely trained on a single dedicated topology, whereas Neuro-DCF-general is obtained with our random graph training method.

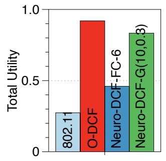
For the evaluation of Neuro-DCF-general, we trained Neuro-DCF-$G(10, 0.3)$ by randomly sampling the graph from $G(10, 0.3)$ every episode, and used this model in all topologies without retraining. Neuro-DCF-single models are trained on each topology, e.g., Neuro-DCF-FIM is trained only on the FIM topology instead of randomized graph inputs. We set the EWMA weight α to 0.9, queue reward weight β to 0.01, discount factor γ to 0.99, and used decaying learning rate from 5×10^{-5} to 0. We use 20-core i9-9900X with 3.50 GHz CPU with 128 GB RAM, and 2 TITAN Xp GPUs for training Neuro-DCF-$G(10, 0.3)$ algorithm. We compare our Neuro-DCFs with two baselines, 802.11 and O-DCF. 802.11 is currently a standardized MAC algorithm up to the latest WiFi 6, and O-DCF is the implementation of optimal CSMA. A detailed description of the hyper-parameters and neural network configuration is provided in technical report [26].

4.3 Results

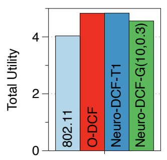
In this section, we present four-fold evaluation metrics with multiple aspects: (i) Average throughput and queue length, (ii) Short-term utility, (iii) Queue dynamics over time, and (iv) generalization performance. Each result shows the average and 95% confidence value of 5 randomly initialized models. Due to the space limitation, we only provide the results on the FIM, FC-6, T1, and T2 topologies. Additional results at each evaluation from variant topologies are provided in technical report [26].



(a) FIM



(b) FC-6



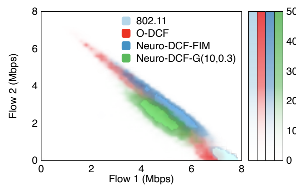
(c) T1

Figure 7: Total utility of each algorithm at each topology.

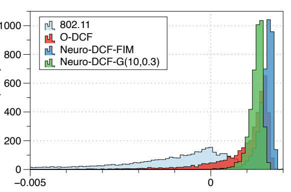
4.3.1 *Throughput and Queue Length.*

The main performance metric we evaluated is the average throughput and delay. The average throughput measures the *long-term behavior*, such as efficiency and fairness, of the MAC algorithm. We are particularly interested in *network utility*, which is denoted as $\sum_i U(\bar{x}^i)$ where $\bar{x}^i$ is the long-term rate of flow i. Since the O-DCF algorithm is guaranteed to achieve the optimal network utility, one of our objectives is to get a comparable performance to O-DCF. The delay metric captures the *short-term behavior*, where the class of optimal CSMA has a particular weakness. We measure the delay by investigating the average queue length, as measuring the raw end-to-end frame delay can be sometimes misleading because of unsent frames. We did not measure the average queue length of vanilla 802.11 because the queue structure in a MAC layer is different from O-DCF and Neuro-DCF, which makes the fair comparison impossible.

Figure 7 shows the total utility of MAC algorithms in each topology. 802.11 has the lowest network utility in all environments. O-DCF and all Neuro-DCF variants show a similar utility, except in FC-6 scenario. This happens because FC-6 has a very harsh interference characteristics, and therefore the short-term estimation of the throughput does not comply with the long-term throughput.



(a) Throughput



(b) Utility

Figure 8: Density map of short-term throughput and histogram of utility in FIM topology.

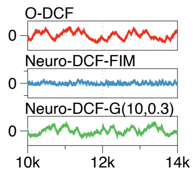
Neuro-DCF-$G(10, 0.3)$ shows a better utility than Neuro-DCF-FC-6 because it can learn the general set of interference structures and effectively find the better solution.

Upper row of Figure 6 shows the average per-flow throughput in various interference topologies. In Figure 6(a), the inner flow is largely starved under the legacy 802.11 DCF algorithm, whereas both O-DCF and Neuro-DCF algorithm achieve the optimal channel allocation. While the long-term utility of O-DCF and Neuro-DCF show similar numbers, bottom row of Figure 6 shows that Neuro-DCF has significantly better queueing performance than O-DCF in all topologies. Neuro-DCF-$G(10, 0.3)$ mostly shows slightly less performance than the Neuro-DCF-single models, which can be considered as the cost of generality.

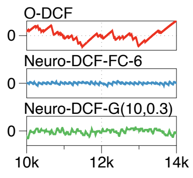
4.3.2 *Short-term Utility.*

Along with long-term throughput and utility measurement, we also conduct an experiment to provide statistics of the short-term throughput, showing how our MAC algorithm works in more details. We collect the short-term throughput and utility numbers by EWMA-estimation, same as (6). Figure 8a shows the two-dimensional histogram of channel allocation between different flows. As can be seen in the figure, the vanilla 802.11 algorithm sometimes fails to share the channel between different flows. O-DCF and Neuro-DCF shares the similar long-term throughput, but the short-term statistics are quite different. The histogram of Neuro-DCF shows more concentrated measurement towards the center which indicates that Neuro-DCF achieves more stable scheduling compared to the fluctuating behavior of the O-DCF.

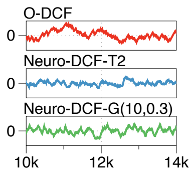
Figure 8b shows the histogram of short-term utility in various network topologies. The short-term utility we use is derived from (5), the EWMA-estimated short-term throughput. The Neuro-DCF algorithms shows the most concentrated utility distribution compared to 802.11 and O-DCF. This means that the short-term network utility varies less in Neuro-DCF, which leads to more stabilized scheduling behavior. Neuro-DCF-$G(10, 0.3)$ shows slightly less concentrated statistics compared to the Neuro-DCF-single models because of the generalization.



(a) FIM topology



(b) FC-6 topology



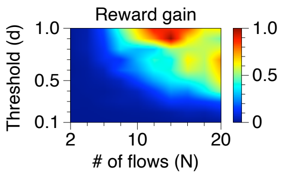
(c) T2 Topology

Figure 9: Queue length deviation from 10k ms to 14k ms of flow 1 of each topology.

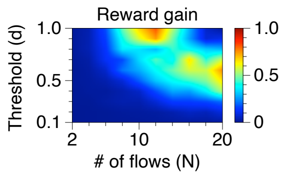
4.3.3 *Queue Dynamics over Time.*

We further characterize the instantaneous behavior of the MAC algorithms by looking at the queue length dynamics over time. The queue length of each flow summarizes the arrival and service process of the scheduling dynamics. Optimal CSMA algorithms have been reported to have a poor performance in terms of queueing dynamics despite being optimal in the long-term.

Figure 9 depicts the queueing dynamics of O-DCF and Neuro-DCF. As can be seen in the figure, there is a relatively large deviation of the queue length in O-DCF and Neuro-DCF-$G(10, 0.3)$, and the Neuro-DCF-single models show almost zero deviation. Note that Figure 9 only shows the deviation from the mean. The Neuro-DCF-single models are trained on a single topology, so the policy does not have to infer its surrounding interference structure. However, O-DCF and Neuro-DCF-$G(10, 0.3)$ have to collect some observations to infer the wireless environment and control accordingly. This makes slightly larger queue length deviations of both algorithms, but Neuro-DCF achieves greatly reduced mean queue length compared to O-DCF. In O-DCF, the sender has to wait the backlog to grow sufficiently large in order to access the channel more aggressively.



(a) Neuro-DCF-G(10,0.3)



(b) Neuro-DCF-T2

Figure 10: (a) and (b) represent max-min normalized reward gain of Neuro-DCF-$G(10, 0.3)$ and Neuro-DCF-T2 compared with O-DCF, respectively.[4]

4.3.4 *Generalization.*

In this section we show the generalization ability of Neuro-DCF training algorithm. So far, we have used Neuro-DCF-$G(10, 0.3)$ to cover various topologies. Aside from synthetic topologies like FIM or FC, the general topologies T1, T2 and T3 are all the instances of $G(10, 0.3)$, therefore we can say that we verified the *in-distribution* generalization performance of Neuro-DCF-$G(10, 0.3)$. In this section, we measure the ability of Neuro-DCF-$G(10, 0.3)$ model on the topologies outside of G(10, 0.3), in order to evaluate the *out-distribution* generalization performance of Neuro-DCF-$G(10, 0.3)$. To see this, we take the trained models and tested them in graph topologies that lie in the (N, d) space of random geometric graphs. In Figure 10, we plot the 10×10 grid heatmap, where each patch of the grid represents the discretized space of N and d, namely $N \in \{2, 4, \cdots, 20\}$ and $d \in \{0.1, 0.2, \cdots, 1.0\}$. Five graph topologies are sampled from each configuration of (N, d), and we also measure the performance of O-DCF in all settings to evaluate the relative improvement of Neuro-DCF. Figure 10a shows the generalization performance of Neuro-DCF-$G(10, 0.3)$, and Figure 10b shows that of Neuro-DCF-T2. According to the figure, the random graph trained algorithm Neuro-DCF-$G(10, 0.3)$ outperforms the Neuro-DCF-single model by showing more improvements in wider span of (N, d) surface.

[4]The max-min normalized gain denotes the normalized difference of the reward in Neuro-DCF and in O-DCF. The maximal difference becomes 1, the minimal becomes 0, and the difference value is linearly scaled accordingly.

5 CONCLUSION

In this paper, we introduced a cooperative MARL approach for addressing the wireless scheduling problem. The wireless MAC has been researched for decades, but the current state-of-the-art algorithms cannot be said to have achieved optimality. Our main objective is to present a learning-based approach for training an efficient wireless MAC controller. With our approach, the delay is minimized compared to the optimal CSMA while preserving the optimality in throughput and fairness. We utilized the modified version of the PPO algorithm for stable and efficient training, and we proposed the novel GNN-based critic architecture to train a generalized MAC algorithm to achieve optimal utility and minimal delay. The new paradigm of experience-driven engineering has therefore shown great possibilities by this demonstration, and we hope to stimulate the research community to practice this discipline and further improve the efficiency of wireless networking.

ACKNOWLEDGMENTS

This work was supported by BK21 FOUR project of the National Research Foundation of Korea Grant. The authors would like to thank Roben D. Delos Reyes for constructive criticism of the manuscript.

REFERENCES

[1] Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. 2020. Deep coordination graphs. In *Proc. of ICML.*

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[3] Frederico Calì, Marco Conti, and Enrico Gregori. 2000. Dynamic tuning of the IEEE 802.11 protocol to achieve a theoretical throughput limit. *IEEE/ACM Transactions on networking* 8, 6 (2000), 785–799.

[4] Ursula Challita, Li Dong, and Walid Saad. 2018. Proactive Resource Management for LTE in Unlicensed Spectrum: A Deep Learning Perspective. *IEEE transactions on wireless communications* 17, 7 (2018), 4674–4689.

[5] Sandeep Chinchali, Pan Hu, Tianshu Chu, Manu Sharma, Manu Bansal, Rakesh Misra, Marco Pavone, and Sachin Katti. 2018. Cellular Network Traffic Scheduling with Deep Reinforcement Learning. In *Proc. of AAAI.*

[6] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Proc. of NeurIPS.*

[7] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual multi-agent policy gradients. In *Proc. of AAAI.*

[8] Piotr Gawłowicz and Anatolij Zubow. 2019. ns-3 Meets OpenAI Gym: The Playground for Machine Learning in Networking Research. In *Proc. of MSWiM.*

[9] Yan Grunenberger, Martin Heusse, Franck Rousseau, and Andrzej Duda. 2007. Experience with an implementation of the Idle Sense wireless access method. In *Proc. of ACM CoNEXT.*

[10] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. 2017. Cooperative multi-agent control using deep reinforcement learning. In *Proc. of AAMAS.*

[11] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. 2017. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183* (2017).

[12] Martin Heusse, Franck Rousseau, Romaric Guillier, and Andrzej Duda. 2005. Idle sense: an optimal access method for high throughput and fairness in rate diverse wireless LANs. In *Proc. of SIGCOMM.*

[13] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proc. of CVPR.*

[14] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *Proc. of ICML.*

[15] Libin Jiang and Jean Walrand. 2009. A distributed CSMA algorithm for throughput and utility maximization in wireless networks. *IEEE/ACM Transactions on Networking* 18, 3 (2009), 960–972.

[16] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. of ICLR.*

[17] Jaewook Kwak, Chul-Ho Lee, and Do Young Eun. 2013. Exploiting the past to reduce delay in CSMA scheduling: A high-order Markov chain approach. *ACM SIGMETRICS Performance Evaluation Review* 41, 1 (2013), 353–354.

[18] Hojin Lee, Sangwoo Moon, and Yung Yi. 2015. A-DCF: Design and implementation of delay and queue length based wireless MAC. In *Proc. of INFOCOM.*

[19] Jinsung Lee, Hojin Lee, Yung Yi, Song Chong, Edward W Knightly, and Mung Chiang. 2015. Making 802.11 DCF Near-Optimal: Design, Implementation, and Evaluation. *IEEE/ACM Transactions on Networking* 24, 3 (2015), 1745–1758.

[20] Husheng Li. 2010. Multiagent Q-Learning for Aloha-Like Spectrum Access in Cognitive Radio Systems. *EURASIP Journal on Wireless Communications and Networking* 2010 (2010), 1–15.

[21] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *Proc. of ICML.*

[22] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *Proc. of ICLR.*

[23] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proc. of NeurIPS.*

[24] Nicholas Mastronarde, Jalil Modares, Changcan Wu, and Jacob Chakareski. 2016. Reinforcement Learning for Energy-Efficient Delay-Sensitive CSMA/CA Scheduling. In *Proc. of GLOBECOM.*

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[26] Sangwoo Moon, Sumyeong Ahn, Kyunghwan Son, Jinwoo Park, and Yung Yi. 2020. *Neuro-DCF: Design of Wireless MAC via Multi-Agent Reinforcement Learning Approach.* Technical Report. http://lanada.kaist.ac.kr/neuro-dcf/neuro-dcf-tech-report.pdf

[27] Navid Naderializadeh, Fan H Hung, Sean Soleyman, and Deepak Khosla. 2020. Graph Convolutional Value Decomposition in Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2010.04740* (2020).

[28] Oshri Naparstek and Kobi Cohen. 2018. Deep Multi-User Reinforcement Learning for Distributed Dynamic Spectrum Access. *IEEE Transactions on Wireless Communications* 18, 1 (2018), 310–323.

[29] Frans A Oliehoek, Christopher Amato, et al. 2016. *A concise introduction to decentralized POMDPs.* Vol. 1. Springer.

[30] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proc. of CVPR.*

[31] George F Riley and Thomas R Henderson. 2010. The ns-3 Network Simulator. In *Modeling and tools for network simulation.* Springer, 15–34.

[32] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. High-dimensional continuous control using generalized advantage estimation. In *Proc. of ICLR.*

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[34] Vasilios A Siris and George Stamatakis. 2006. Optimal CWmin selection for achieving proportional fairness in multi-rate 802.11 e WLANs: test-bed implementation and evaluation. In *Proc. of WiNTECH.*

[35] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

[36] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proc. of NeurIPS.*

[37] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. 2017. Multiagent cooperation and competition with deep reinforcement learning. *PloS one* 12, 4 (2017), e0172395.

[38] Justin K Terry, Nathaniel Grammel, Ananth Hari, Luis Santos, Benjamin Black, and Dinesh Manocha. 2020. Parameter Sharing is Surprisingly Useful for Multi-Agent Deep Reinforcement Learning. *arXiv preprint arXiv:2005.13625* (2020).

[39] Nelson Vithayathil Varghese and Qusay H Mahmoud. 2020. A survey of multi-task deep reinforcement learning. *Electronics* 9, 9 (2020), 1363.

[40] Shangxing Wang, Hanpeng Liu, Pedro Henrique Gomes, and Bhaskar Krishnamachari. 2018. Deep Reinforcement Learning for Dynamic Multichannel Access in Wireless Networks. *IEEE Transactions on Cognitive Communications and Networking* 4, 2 (2018), 257–265.

[41] Ajit Warrier, Sankararaman Janakiraman, Sangtae Ha, and Injong Rhee. 2009. DiffQ: Practical differential backlog congestion control for wireless networks. In *Proc. of INFOCOM.*

[42] Cathy Wu, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M Bayen, Sham Kakade, Igor Mordatch, and Pieter Abbeel. 2018. Variance reduction for policy gradient with action-dependent factorized baselines. In *Proc. of ICLR.*

[43] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. 2018. Experience-driven Networking: A Deep Reinforcement Learning based Approach. In *Proc. of INFOCOM.*

[44] Se-Young Yun, Yung Yi, Jinwoo Shin, and Do Young Eun. 2012. Optimal CSMA: A Survey. In *Proc. of ICCS.*