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

Received 22 November 2022; revised 30 March 2023 and 6 June 2023; accepted 16 June 2023.
Date of publication 20 June 2023; date of current version 30 June 2023.

The associate editor coordinating the review of this article and approving it for publication was Y. Zhang.

Digital Object Identifier 10.1109/TMLCN.2023.3288090

A Three-Tier Deep Learning-Based Channel Access Method for WiFi Networks

YIWEI HUANG AND KWAN-WU CHIN

School of Electrical, Computer and Telecommunications Engineering, University of Wollongong, Wollongong, NSW 2522, Australia
CORRESPONDING AUTHOR: Y. HUANG (yh733@uowmail.edu.au)

ABSTRACT Future WiFi networks require a channel access method that provides users with high capacity. Such a method must consider 1) channel bonding, which improves the transmission capacity of Access Points (APs); and 2) spatial reuse, where APs tune their Clear Channel Assessment (CCA) threshold and transmit power in order to transmit concurrently with neighboring APs. To date, there are no solutions that *jointly* optimize the channels used by an AP, and the CCA threshold and transmit power of a bonded channel. To this end, we outline a three-tier deep learning approach. Briefly, at Layer-1, it selects a set of transmitting channels. At Layer-2 and Layer-3, it respectively determines the transmit power and CCA threshold for each selected channel. An AP then employs deep reinforcement learning to learn the optimal policy for each layer given its interference intensity and queue length. The simulation results show that when compared to three competing solutions, an AP that uses our approach is able to reduce its queue length by up to 62.52% under realistic traffic load.

INDEX TERMS Medium access, capacity, Markov decision process, interference, channel aggregation.

I. INTRODUCTION

IEEE 802.11 based Wireless Local Area Networks (WLANs), aka WiFi networks, play an essential role in people's daily activities. Indeed, Access Points (APs) are ubiquitous and densely deployed in places such as offices, shopping malls, stadiums and airports [1]. These APs must support a high number of users. As an example, in the sports event studied in [2], WiFi networks need to provide services to 12,000 users simultaneously with a maximum aggregated data rate of 3.5 Gbps. However, with limited spectrum resources, densely deployed APs may experience significant interference from neighboring cells if they operate on the same channel [3]. Moreover, emerging Internet applications such as online meetings/education, ultra high definition streaming and virtual reality videos require high throughput or capacity to ensure a high quality of service [4].

There are two methods to improve network capacity. One method is via channel bonding or aggregation [5], which allows an AP to combine multiple channels together to form a wide bandwidth. For example, an IEEE 802.11ax AP is able to bond up to eight 20 MHz channels to form a 160 MHz channel. As shown in [2], with increasing bandwidth, an AP is able to transmit with a high data rate and thus, improves its capacity.

Another method is to maximize spatial reuse, meaning multiple nodes are able to transmit concurrently [6]. The level of spatial reuse is determined by the adopted Medium Access Control (MAC) scheme. Specifically, current WiFi networks rely on Carrier Sense Multiple Access with collision avoidance (CSMA/CA) for channel access. Each device uses a Clear Channel Assessment (CCA) threshold to determine the status of a channel. If the power level on a channel falls below a given CCA threshold, a device is allowed to transmit its packets. Otherwise, the channel is not available. Hence, a key problem is to determine a suitable threshold that allows multiple APs or/and users to transmit concurrently without causing too much interference to one another [7]. In this respect, Transmit Power Control (TPC) is essential [1], whereby an AP adjusts its transmit power to minimize interference caused to neighboring APs [5] or/and avoid triggering the CCA threshold of these APs, which causes them to defer their transmission unnecessarily.

To illustrate the previous points, consider Fig. 1 where an AP serves four users. The AP is able to bond up to three channels for transmissions, and all users and the AP experience

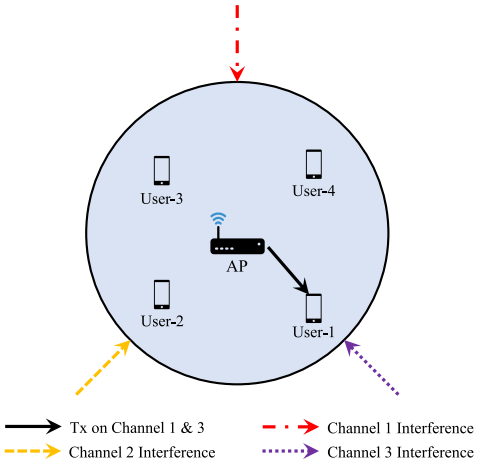


FIGURE 1. An example WLAN. The AP is able to bond up to three channels and use different transmit power on each channel. Both AP and user experience interference from neighboring cells on each channel.

inter-cell interference on different channels. Assume the AP has packets for User-1. Further, it experiences high interference on Channel-2. In addition, assume User-1 experiences low interference on Channel-1. In this regard, the AP can choose to bond Channel-1 and Channel-3, and increase the CCA threshold on these channels to gain more opportunities to transmit. Moreover, the AP can allocate a higher transmit power on Channel-3 as compared to Channel-1 if the Signal to Interference plus Noise Ratio (SINR) on Channel-1 exceeds a given threshold. This helps to ensure the SINR on both channels exceeds a given value, and reduce the interference to neighboring cells. Consequently, the AP is able to transmit packets to User-1 with a high data rate.

The use of channel bonding, transmit power control and CCA threshold adjustment creates several problems. In particular, an AP that bonds multiple channels may suffer significant interference from neighboring cells [8]. Moreover, it leads to a lower power density, i.e., Watts/Hz [9] which may lead to a low data rate. Secondly, although transmit power control may reduce the interference to neighboring APs, it may also result in a low signal strength. In this respect, if an AP uses a low transmit power and transmits over a bonded channel, neighboring devices may not hear its transmission on some channels and thereby start to transmit [10]. This may cause severe interference to ongoing transmissions or even collisions. Moreover, if the CCA threshold is low, an AP may transmit even when there is high interference, which degrades its transmission capacity. Consequently, an AP needs to optimize its channel bonding strategy, transmit power and CCA threshold carefully. Otherwise, the AP may experience significant delays or queue overflows.

There are three challenges to consider when jointly optimizing channel bonding, CCA threshold and transmit power. First, the channel conditions on each channel vary over time. Next, an AP may bond different combination of channels for each transmission over time. Lastly, the traffic arrival at an AP is random.

Henceforth, this paper makes the following contributions:

- It addresses a novel problem that calls for a solution to optimize an AP's channel bonding policy, transmit power and CCA threshold under random traffic and channel conditions. Further, it considers both adjacent and non-adjacent channel bonding. The AP's aim is to maximize its throughput and minimize its queue length.
- It presents the first formal model of the said problem. Specifically, this paper formulates a three-layer Markov Decision Process (MDP) for the said problem [11], and outlines a three-tier learning approach based on Deep Q-Network (DQN) [12] and Deep Deterministic Policy Gradient (DDPG) [13] that runs on an AP to independently solve the MDP. The proposed three-tier approach only requires local information, such as an AP's current queue length and locally measured interference on each channel, to learn the optimal policy. This means the proposed approach scales with network size and it is decentralized. Further, the proposed approach *does not* require prior knowledge of an environment, meaning it is model-free. We emphasize that our approach is run by a single AP and *does not* require an AP to cooperate with neighboring APs, which may be managed by different entities. Lastly, our work is significant because the amount of traffic and interference vary over time, which motivates the use of learning based solutions such as DDPG.
- To the best of our knowledge, there are no prior works that jointly and adaptively optimize channel bonding, transmit power and CCA threshold for APs with random traffic arrivals in order to improve network capacity, see Section II for details. Hence, this paper is the first to outline a solution for the said novel problem.
- This paper contains the first study on the aforementioned problem and solution. The simulation results show that an AP running the proposed three-tier learning approach is able to reduce its queue length by 32.94%, 50.99% and 62.52% as compared to algorithms that use fixed or randomly strategies on channel bonding, transmit power control and CCA threshold.

The rest of this paper is organized as follows. Section II compares prior works that consider channel bonding, transmit power control, CCA threshold and works that apply Reinforcement Learning (RL). Section III presents our system model and problem, and Section IV outlines a brief background of MDP, DQN and DDPG. After that, Section V details the proposed three-tier learning approach. Lastly, Section VI presents simulation results, and Section VII concludes the paper.

II. RELATED WORKS

A number of works have considered applying channel bonding to wireless networks. For example, the work in [14], [15],

TABLE 1. Comparison of previous works.

Paper	CB	TPC	CCA	RL
[14]–[23]	✓	✗	✗	✗
[10], [24]–[28]	✓	✗	✗	✓
[29], [30]	✓	✓	✗	✓
[31], [32]	✓	✗	✓	✗
[33], [34]	✗	✓	✓	✓
Our work	✓	✓	✓	✓

[16], [17], [18], and [19] assigns non-overlapping channels to each AP. The aim is to reduce interference and satisfy the traffic demands of APs. The study in [20], [21], and [22] uses game theory *or* integer nonlinear programming to determine a channel bonding strategy for each AP. Their aim is to improve overall network throughput. The work in [23] determines channel bonding policies by jointly monitoring the traffic load on secondary channels as well as the delay experienced by all users. Its aim is to minimize the transmission delay for each user.

Many works have applied machine learning techniques to optimize channel bonding strategies. For example, the work in [24] uses deep learning to optimize the probability of selecting a set of channels. In [10], the authors use RL to select a set of channels to bond taking into consideration the interference on each channel. Their goal is to maximize the throughput of each AP. The work in [25] uses multi-agent RL and considers the traffic load of each AP. Agents learn to select a primary channel and a number of secondary channels to bond. The aim is to minimize the average transmission delay. The work in [26] aims to minimize interference between APs and satisfy the time varying traffic demands at each AP. The AP in [27] runs RL to learn the optimal probability to sense and bond each secondary channel. Its goal is to maximize the total network throughput given some traffic load. The authors in [28] first use a Markov chain to model the throughput for bonded channels before proposing a multi-arm bandit algorithm to determine a set of channels to bond and avoid collisions.

There are also works that consider combining channel bonding with transmit power control. In [29], the authors use RL to jointly optimize channel bonding and transmit power of an AP. The goal is to maximize the energy efficiency of an AP with random traffic arrivals. The work presented in [30] first uses Q-learning on each femtocell to select a set of channels to transmit. After that, the authors use convex optimization to allocate transmit power on each selected channel, and they aim to maximize the throughput of femtocells as well as minimize the interference experienced by macrocell users.

Some other works focus on incorporating channel bonding with CCA threshold adjustment. For example, the work in [31] and [32] adjusts the CCA threshold of APs first. In particular, the authors of [31] first determine the CCA threshold on the primary channel of each AP by jointly considering average interference, signal strength to associated users and channel occupancy time. After that, the CCA threshold on each secondary channel is obtained by adding a fixed value to the CCA threshold on the primary channel. The work in [32] adjusts the CCA threshold for secondary channels only. The CCA threshold on each secondary channel is the same, and is calculated based on a SINR threshold and the distance between an AP and users. Then, the work in [31] and [32] bonds channels for each AP according to CCA results. In addition, there are some other works that jointly optimize the transmit power and CCA threshold over a channel. For example, the work in [33] and [34] uses reinforcement learning to assign a primary channel, transmit power or CCA threshold to each AP. Their aim is to maximize the throughput of each AP.

Our work is fundamentally different to prior works, see Table 1 for a comparison. Firstly, although previous works consider channel bonding, transmit power control *or* CCA threshold adjustment, e.g., [14], [15], [19], [29], [30], [31], [32], [33], and [34], they *do not* jointly optimize them to improve spectrum efficiency. Secondly, works such as [14], [15], [19], and [22] use a centralized method to optimize channel bonding. These methods require global information and cooperation between APs. In contrast, we consider a decentralized solution, where an AP independently determines its policy. Further, our method is model-free and only requires local information, i.e., the current queue length and experienced interference. In addition, previous works, e.g., [14], [15], [16], [17], and [18], only bond adjacent channels. However, we consider bonding both adjacent and non-adjacent channels; this provides more flexibility than bonding adjacent channels and it is now supported in IEEE 802.11ax [5]. Moreover, although the work in [31] and [32] adjusts CCA threshold, it considers assigning the same CCA threshold to all channels. In contrast, we adapt the CCA threshold on each channel. Lastly, research such as [33] does not consider traffic at APs, and the work in [10] considers a fixed channel gain. In contrast, we consider time-varying channel gains and traffic arrivals.

III. SYSTEM MODEL

Time is divided into T time slots and indexed by t; each time slot has length δ (in seconds). We consider an AP i with a set of users $\mathcal{U}$. Let d_{iu} denote the Euclidean distance between AP i and a user $u \in \mathcal{U}$. There are N channels in the set $\mathcal{C}$; each channel has a fixed bandwidth of B MHz. In each time slot t, AP i transmits to a user u over a set of channels $\mathcal{C}_i^t \subseteq \mathcal{C}$. We assume the interference experienced by AP i and user u is generated by a set of neighboring APs, denoted as $\mathcal{N}_i$; this set contains any APs that may degrade the SINR of AP i and user u. Further, we denote by $\mathcal{C}_j^t$ the set of channels used by a neighboring AP $j \in \mathcal{N}_i$ in time slot t. Table 2 lists our notations.

AP i uses CSMA/CA for channel access where the CCA threshold of channel c at time t is denoted as γ_c^t. Moreover,

TABLE 2. Parameter values.

Notation	Explanation
i	AP i
$\mathcal{U}$	The set of users associated with AP i
$\mathcal{C}$	The set of channels
$\mathcal{C}_i^t$	The set of channels used by AP i in time slot t
$\mathcal{N}_i$	The set of neighboring APs of AP i
$\mathcal{N}_c^t$	The set of neighboring APs that are transmitting on channel c in time slot t
B	The bandwidth of a single channel (in MHz)
T	Total number of time slots
δ	Length of each time slot (in seconds)
d_{iu}	The Euclidean between AP i and user u
g_{iu}^t	The channel gain from AP i to user u in time slot t
I_{uc}^t	The interference experienced by user u on channel c in time slot t
N_b	The ambient noise power density (in Watt/Hz)
β_{uc}^t	SINR from AP i to user u on channel c in time slot t
r_c^t	The data rate of AP i on channel c in time slot t
r_i^t	The aggregated data rate of AP i in time slot t
γ_c^t	The CCA threshold of AP i on channel c in time slot t
P_{ic}^t	The transmit power of AP i on channel c in time slot t
q_i^t	The queue length of AP i in time slot t
λ_i^t	The number of packets arriving at AP i in time slot t
L	Packet size (in bits)

let $\mathcal{N}_i^t \subseteq \mathcal{N}_i$ be the set of transmitting neighboring APs at the start of time slot t. Further, we denote transmitting APs on channel c at time t as $\mathcal{N}_c^t = \{j | j \in \mathcal{N}_i^t \wedge c \in \mathcal{C}_j^t\}$.

We consider block fading, meaning the channel gain is fixed within one time slot and differs across time slots. The channel power gain from AP i to user u in time slot t is denoted as g_{iu}^t, and it is calculated using the Log-distance path loss model (in dB) [35]:

$$\mathrm{PL}(d_{iu}) = \mathrm{PL}(d_0) + 10\omega \log_{10}\left(\frac{d_{iu}}{d_0}\right) + X_g, \tag{1}$$

where $\mathrm{PL}(d_0)$ is the reference path loss (in dB) measured at reference distance d_0, and ω is the path loss exponent. The term X_g (in dB) is a random variable drawn from a zero-mean Gaussian distribution $\mathcal{N}(0, \sigma^2)$, representing shadowing effect. Then, the channel power gain is $g_{iu}^t = \frac{1}{10^{\mathrm{PL}(d_{iu})/10}}$.

We denote by β_{uc}^t the SINR of a transmission from AP i to user u over channel c in time slot t. Formally,

$$\beta_{uc}^t = \frac{P_{ic}^t g_{iu}^t}{I_{uc}^t + N_b B}, \tag{2}$$

where N_b is the ambient noise power density (in Watt/Hz), and P_{ic}^t is the transmit power (in Watt) used by AP i on channel c in time slot t. Note that the transmit power satisfies $0 \le P_{ic}^t \le P_{max}$ and $\sum_{c \in \mathcal{C}_i^t} P_{ic}^t = P_{max}$. The term I_{uc}^t is the aggregated interference experienced by user u on channel c in time slot t, which is calculated as

$$I_{uc}^t = \sum_{j \in \mathcal{N}_c^t} P_{jc}^t g_{ju}^t. \tag{3}$$

We denote by r_c^t the theoretical data rate of AP i on channel c in time slot t. This data rate is calculated using the Shannon-Hartley formula, which is given by

$$r_c^t = B \log_2(1 + \beta_{uc}^t). \tag{4}$$

Then, the aggregated data rate r_i^t of AP i over $\mathcal{C}_i^t$ channels in time slot t is given by

$$r_i^t = \sum_{c \in \mathcal{C}_i^t} r_c^t. \tag{5}$$

Note that in this paper we consider the theoretical capacity of a channel given its SINR. In practice, the data rate is determined by the Modulation and Coding Scheme (MCS) adopted by a sender. To this end, the value of r_c^t can be set to the highest possible MCS or data rate for a given SINR.

We assume AP i has a queue of packets to transmit. The length of the queue at the end of time slot t is q_i^t, where $0 \le q_i^t \le q_{max}$. At the beginning of each time slot t, we use λ_i^t to denote the number of packets arriving at AP i, where the value of λ_i^t is sampled from a probability distribution. Further, we assume each packet has a fixed size of L bits. Define $\Gamma_i^t \in \{0, 1\}$ to indicate whether AP i transmits in time slot t ($\Gamma_i^t = 1$); otherwise, we have $\Gamma_i^t = 0$. The queue length at AP i evolves as per

$$q_i^t = \min\left(\max\left(q_i^{t-1} + \lambda_i^t - \Gamma_i^t \frac{r_i^t \delta}{L}, 0\right), q_{max}\right). \tag{6}$$

Let π be a policy used by AP i that selects a set of channels $\mathcal{C}_i^t$, and assigns transmit power P_{ic}^t on each channel $c \in \mathcal{C}_i^t$ at the beginning of time slot t. Moreover, the policy π also adjusts the CCA threshold $\gamma_c^t \in [\gamma_{min}, \gamma_{max}]$ on each channel $c \in \mathcal{C}_i^t$. Define by $R(\pi)$ an objective function for policy π. Formally, we have

$$R(\pi) = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}^{\pi}\left[\sum_{t=1}^{T} \left(\eta_1 r_i^t - \eta_2 q_i^t\right)\right], \tag{7}$$

where η_1 and η_2 are two weights that balance the data rate and queue length of AP i, and $\mathbb{E}^{\pi}[.]$ refers to the expectation over the objective value when using policy π. We note that the weight η_2 can be revised to be a non-linear function of AP i's queue length, whereby a high penalty is recorded if the queue length is at maximum.

Let Ω be a collection of policy π. The problem at hand is to find the optimal policy π^* that maximizes the objective function $R(\pi)$ over time. Mathematically, we have

$$\pi^* = \arg\max_{\pi \in \Omega} R(\pi). \tag{8}$$

IV. A MARKOV DECISION PROCESS MODEL

We first discuss Markov Decision Process (MDP) [11]. After that, we introduce DQN [12] and DDPG [13]. Lastly, we introduce the simplex sampling method [36], which is used to sample possible transmit power allocations over one or more channels.

A. MARKOV DECISION PROCESS

An MDP is defined as a tuple with four elements $(\mathcal{S}, \mathcal{A}, \mathcal{R}(s_t, a_t), \mathcal{P}(s_{t+1}|s_t, a_t))$, where $\mathcal{S}$ and $\mathcal{A}$ denote the set of states and actions, respectively. In each time slot t, an agent, i.e., AP i in our problem, observes a state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$. The environment then returns a reward $\mathcal{R}(s_t, a_t)$, and moves from state s_t to $s_{t+1} \in \mathcal{S}$ with probability $\mathcal{P}(s_{t+1}|s_t, a_t)$. Let $\pi(s_t)$ be a policy used by an agent, where the policy outputs an action a_t given state s_t, i.e., $a_t = \pi(s_t)$. Let $V^{\pi}(s)$ be a value function that measures the expected long-term reward that starts from state s using policy π thereafter. Mathematically, we have

$$V^{\pi}(s) = \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty} \gamma^{t+k} \mathcal{R}(s_{t+k}, \pi(s_{t+k}))|s_t = s\right], \quad (9)$$

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

The goal of an agent is to find the *optimal policy* π^* that maximizes the value function for all states, denoted by V^*. This optimal value function V^* can be computed using Bellman's equation [11] as

$$V^*(s_t) = \max_{a_t \in \mathcal{A}} \Bigg[\mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1}|s_t, a_t) V^*(s_{t+1})\Bigg], \quad (10)$$

where γ is the discount factor. The optimal policy π^* is then given by

$$\pi^*(s_t) = \arg\max_{a_t \in \mathcal{A}} \Bigg[\mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1}|s_t, a_t) V^*(s_{t+1})\Bigg]. \quad (11)$$

B. DEEP Q-NETWORK

DQN is a value based reinforcement learning algorithm [12]. It learns the optimal policy by approximating the optimal Q-value for each state-action pair [12]. Therefore, DQN supports discrete actions, e.g., selecting a set of channels. DQN consists of two neural networks, an evaluation network θ and a target network θ'. The two networks have the same structure, where the evaluation network θ outputs a Q-value for a given state-action pair, denoted as $\mathcal{Q}(s_t, a_t; \theta)$, and the target network θ' outputs the corresponding target Q-value $\mathcal{Q}(s_t, a_t; \theta')$.

DQN uses experience replay to update the weights of its neural networks. Specifically, each combination of state, action, reward and next state $(s_t, a_t, \mathcal{R}(s_t, a_t), s_{t+1})$ is called an *experience*. DQN will store an *experience* in each time slot into its memory buffer $\mathcal{M}$, which stores up to $|\mathcal{M}|$ experiences. For every K time slots, DQN uniformly samples a batch of experiences from $\mathcal{M}$ to update the weights of its evaluation network θ. The goal is to minimize a loss function which is given by

$$\mathcal{L}(\theta) = \mathbb{E}[(\mathbf{y} - \mathcal{Q}(s_t, a_t; \theta))^2], \quad (12)$$

where

$$\mathbf{y} = \mathcal{R}(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} \mathcal{Q}(s_{t+1}, a; \theta'). \quad (13)$$

In addition, to ensure stability, for every K' time slots, the weights of the target network θ' are replaced by the weights of the evaluation network θ.

C. DEEP DETERMINISTIC POLICY GRADIENT

A drawback of DQN is that it is not able to learn the optimal policy when the action space is continuous [13], e.g., the transmit power and CCA threshold of AP i. Therefore, we employ DDPG, an actor-critic based algorithm, to address the said issue [13]. DDPG has four neural networks, namely an actor network θ^{μ}, a target actor network $\theta^{\mu'}$, a critic network $\theta^{\mathcal{Q}}$ and a target critic network $\theta^{\mathcal{Q}'}$. The structure of a target actor network and target critic network is the same as the corresponding actor and critic network, respectively. The actor network chooses a deterministic action a_t at each state s_t, denoted as $a_t = \mu(s_t; \theta^{\mu})$, and the critic network evaluates the Q-value $\mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}})$ for each selected action a_t at state s_t. Similarly, the target actor network selects a target action $\mu(s_t; \theta^{\mu'})$, and the target critic network outputs a target Q-value $\mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}'})$.

DDPG also uses experience replay to update the weights of its networks. In particular, for every K time slots, DDPG first samples a batch of experiences from its memory buffer $\mathcal{M}$ to update the weights of its critic network $\theta^{\mathcal{Q}}$. The update follows a similar process as DQN, which aims to minimize the loss function as per

$$\mathcal{L}(\theta^{\mathcal{Q}}) = \mathbb{E}[(\mathbf{y}^{\mathcal{Q}} - \mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}}))^2], \quad (14)$$

where

$$\mathbf{y}^{\mathcal{Q}} = \mathcal{R}(s_t, a_t) + \gamma \mathcal{Q}(s_{t+1}, \mu(s_{t+1}; \theta^{\mu'}); \theta^{\mathcal{Q}'}). \quad (15)$$

Next, the weights of the actor network are updated using the Q-values evaluated by the critic network. Specifically, DDPG first calculates the gradient of Q-values with respect to all actions in sampled batch, denoted as $\nabla_a \mathcal{Q}(s, a; \theta^{\mathcal{Q}})|_{s=s_t, a=\mu(s_t;\theta^{\mu})}$. Further, DDPG calculates the gradient of all actions with respect to the weights of the actor network θ^{μ}, denoted as $\nabla_{\theta^{\mu}} \mu(s; \theta^{\mu})|_{s=s_t}$. Then, by applying the chain rule, the weights of the actor network are updated using a policy gradient method [37] with the following approximation

$$\nabla_{\theta^{\mu}} \mathcal{Q} \approx \mathbb{E}\left[\nabla_a \mathcal{Q}(s, a; \theta^{\mathcal{Q}})|_{s=s_t, a=\mu(s_t;\theta^{\mu})} \nabla_{\theta^{\mu}} \mu(s; \theta^{\mu})|_{s=s_t}\right]. \quad (16)$$

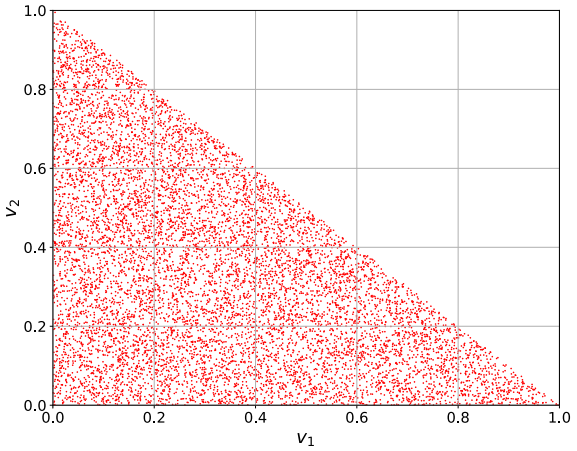


FIGURE 2. An example with 10,000 points that are sampled from a 3D space using simplex [36].

Finally, DDPG applies a soft update on target networks. For every K time slots, the weights of corresponding target networks are updated as per

$$\theta^{Q'} = \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \tag{17}$$
$$\theta^{\mu'} = \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}, \tag{18}$$

where τ is a small positive number, representing the target network update rate.

D. SIMPLEX SAMPLING

A key issue for DDPG is that it needs to randomly select an action to explore the action space. Recall that for the set of channels $\mathcal{C}_i^t$ used by AP i in time slot t, its transmit power allocation is obtained from a $|\mathcal{C}_i^t|$-dimensional space. Further, any transmit power allocation must satisfy $0 \le P_{ic}^t \le P_{max}$ and $P_{max} = \sum_{c\in\mathcal{C}_i^t} P_{ic}^t$. To this end, we apply the simplex sampling method from [36] to determine a transmit power allocation over the set of channels $\mathcal{C}_i^t$.

Let *Simplex*(.) return a vector v with $|\mathcal{C}_i^t|$ elements, i.e., $v = Simplex(|\mathcal{C}_i^t|)$. Each element in v is in the range [0, 1], representing a fraction of the maximum transmit power P_{max} that is used on a certain channel. The function *Simplex*(.) first randomly generates a sequence of values, which it records in the vector $x = \{x_1, x_2, \ldots, x_{|\mathcal{C}_i^t|-1}\}$; each element in x is uniformly sampled from the range [0, 1]. Then, it sorts the elements in x in an increasing order, and adds $x_0 = 0$ and $x_{|\mathcal{C}_i^t|} = 1$ to the beginning and the end of x, respectively. After that, the vector v is obtained based on x, where the i-th value v_i in vector v is calculated as $v_i = x_i - x_{i-1}$. As an example, assume there are three bonded channels. Then the corresponding vector $v = [v_1, v_2, v_3]$ is sampled from a three-dimensional simplex space, i.e., $v = Simplex(3)$. Fig. 2 shows the results of 10,000 samples generated by *Simplex*(3), where each red point (v_1, v_2) represents a sampled vector v. Note that value v_3 is not shown as it can be calculated via $v_3 = 1 - v_1 - v_2$ [36].

V. A THREE-TIER LEARNING APPROACH

Our approach has three layers. This allows us to optimize each system parameter independently whilst keeping the other parameters fixed. For example, for a given set of channels and transmit power, we can then learn a policy to set the CCA threshold for different system states that occur when an AP uses the given number of channels and transmit power. We note that an alternative solution is to jointly optimize the number of channels, transmit power and CCA threshold simultaneously. However, this approach results in a very large action space that leads to low learning efficiency [38]. This problem is further exacerbated by the fact that both transmit power and CCA threshold have continuous values. To this end, our approach decomposes the action space, where each layer optimizes a specific system parameter, and advantageously, it can be optimized using a learning method that is suited to handle discrete or continuous action space.

Next, we first formulate our problem as a three-layer MDP. After that, we show how an AP or agent uses a three-tier learning approach to determine its policy.

A. THREE-LAYER MDP

AP i runs as an agent, operating in an environment with three layers as shown in Fig. 3; each layer corresponds to a task for AP i, and is modeled as an MDP. Briefly, AP i first selects a set of channels $\mathcal{C}_i^t$ in Layer-1. Then, it assigns a transmit power P_{ic}^t for each channel $c \in \mathcal{C}_i^t$ in Layer-2. After that, in Layer-3, AP i selects a CCA threshold γ_c^t for each channel $c \in \mathcal{C}_i^t$. We note that the AP runs each layer sequentially; i.e., each layer is run after receiving an input from a higher layer or a reward from a lower layer.

Referring to Fig. 3, AP i interacts with its environment as follows. In each time slot t, AP i observes a state in each layer. Specifically, the state of Layer-1 is observed from the environment, and the state of Layer-2 and Layer-3 is obtained from Layer-1 and Layer-2, respectively. Based on its observed state, the agent at each layer outputs an action. The AP executes the action of each layer, which yields a reward and a new state for Layer-1.

1) DEFINITIONS

Here, we define the state, action and reward of each layer. Formally, they are as follows:

- Channel Selection (Layer-1):
  - **State** s_t^1: The Layer-1 state s_t^1 consists of the current queue length q_i^{t-1}, and the interference experienced by AP i on each channel $\{I_{ic}^t | c \in \mathcal{C}\}$. Formally, $s_t^1 = \{I_{ic}^t | c \in \mathcal{C}\} \cup \{q_i^{t-1}\}$. Note that AP i can use IEEE 802.11k to collect interference information from associated clients/stations.
  - **Action** a_t^1: The action for Layer-1 is to select a set of channels for AP i to transmit, i.e., $a_t^1 = \mathcal{C}_i^t$, where $\mathcal{C}_i^t \subseteq \mathcal{C}$. For simplicity, we use a_t^1 to represent the set of selected channels $\mathcal{C}_i^t$ in the rest of this section.

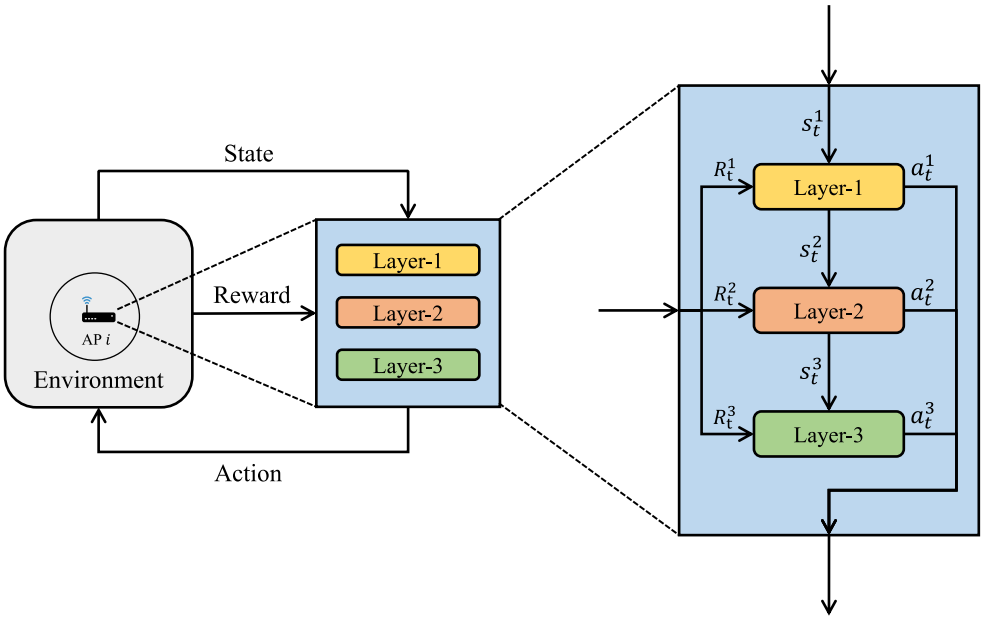


FIGURE 3. **A flowchart of our three-layer MDP model. An agent observes state s_t^1, s_t^2 and s_t^3 for layer-1, layer-2 and layer-3, respectively. For each state, the agent outputs an action a_t^1, a_t^2 and a_t^3 for Layer-1, Layer-2 and Layer-3, respectively. The three actions a_t^1, a_t^2 and a_t^3 are then executed by the agent or AP, which yields the reward and a new state.**

  - **Reward** R_t^1: The reward for Layer-1 is calculated based on the data rate r_i^t and queue length q_i^t of AP i, and is defined as $R_t^1 = \eta_1 r_i^t - \eta_2 q_i^t$, where η_1 and η_2 are two weights.
- Transmit Power Allocation (Layer-2):
  - **State** s_t^2: The state for Layer-2 includes the interference on each channel and the set of channels $\mathcal{C}_i^t$ selected by Layer-1. Here, we define a binary indicator function $b_t^c \in \{0, 1\}$ to track whether channel c is selected in time slot t. That is, the binary indicator function b_t^c returns a value of one if channel c is selected by Layer-1; otherwise it returns zero. Formally, it is defined as

$$b_t^c = \begin{cases} 1, & \text{if } c \in a_t^1, \\ 0, & \text{Otherwise.} \end{cases} \tag{19}$$

    The state for Layer-2 is thus defined as $s_t^2 = \{(I_{ic}^t, b_t^c) \mid c \in \mathcal{C}\}$.
  - **Action** a_t^2: The action for Layer-2 is to assign a transmit power on each selected channel. Formally, $a_t^2 = \{P_{ic}^t | c \in a_t^1\}$. Note that the transmit power P_{ic}^t on each channel satisfies $0 \le P_{ic}^t \le P_{max}$ and $P_{max} = \sum_{c \in a_t^1} P_{ic}^t$.
  - **Reward** R_t^2: The reward for Layer-2 is the same as Layer-1. Formally, we have $R_t^2 = \eta_1 r_i^t - \eta_2 . q_i^t$.
- CCA Threshold (Layer-3):
  - **State** s_t^3: The state for Layer-3 is the transmit power assigned on each channel $c \in a_t^1$. Formally, we have $s_t^3 = a_t^2$.
  - **Action** a_t^3: The corresponding action for Layer-3 is to select a CCA threshold for each channel. Formally, we have $a_t^3 = \{\gamma_c^t | c \in a_t^1\}$.
  - **Reward** R_t^3: The reward for Layer-3 is the achieved data rate of AP i, i.e., $R_t^3 = r_i^t$.

We emphasize that as we propose a model-free approach for practical reason, meaning the transition probability $\mathcal{P}(.)$ in each layer is unknown. Advantageously, our approaches/algorithms allow an AP to learn the optimal policy for different environments upon deployment. Specifically, the AP is only required to observe the states, e.g., varying traffic or interference level, of its environment, and optimize its policy as per our algorithms to determine an action that maximizes its average reward.

B. ALGORITHM DETAILS

We now outline the algorithms used in each tier or layer; these algorithms are run on the same AP. Note that our approach is designed to be run by a single AP. It *does not* require nor assume other APs run our approach. Briefly, an AP, say i, uses DQN to select a set of channels in Layer-1, and uses DDPG to assign a transmit power over each selected channel in Layer-2. For Layer-3, we assume each channel is managed by an agent using DDPG, where the agent on channel c assigns a CCA threshold for the channel. We emphasize that these agents *do not* cooperate with each other, as channels are orthogonal and hence the data rate on a given channel is not a function of other channels.

Algorithm-1 shows the steps of our approach. Initially, in Layer-1, AP i observes the state s_t^1 from its environment,

Algorithm 1 Three-tier learning approach.

Initialize: $\theta_1, \theta_1', \mathcal{M}_1, \epsilon_1$ for DQN in Layer-1.
Initialize: $\theta_2^{\mu}, \theta_2^{\mu'}, \theta_2^{Q}, \theta_2^{Q'}, \mathcal{M}_2, \epsilon_2$ for DDPG in Layer-2.
Initialize: $\{\theta_c^{\mu}, \theta_c^{\mu'}, \theta_c^{Q}, \theta_c^{Q'}, \mathcal{M}_c, \epsilon_c | c \in \mathcal{C}\}$ for DDPG in Layer-3.
1 **while** t = *1,2,…,T* **do**
2 Observe s_t^1
3 $[a_t^1, s_t^2]$ = Layer1SelectChannels$(s_t^1, \theta_1, \epsilon_1)$
4 $[a_t^2, s_t^3]$ = Layer2AssignPower$(s_t^2, \theta_2^{\mu}, \epsilon_2)$
5 **for** *each* $c \in a_t^1$ **do**
6 Get transmit power P_{ic}^t on channel c
7 a_t^c = Layer3AdjustCCA$(P_{ic}^t, \theta_c^{\mu}, \epsilon_c)$
8 **end for**
9 $a_t^3 = \{a_t^c \mid c \in a_t^1\}$
10 Execute a_t^1, a_t^2, a_t^3 and observe R_t^1, R_t^2, R_t^3
11 **if** $t \geq 2$ **then**
12 Store $(s_{t-1}^1, a_{t-1}^1, R_{t-1}^1, s_t^1)$ into $\mathcal{M}_1$
13 Store $(s_{t-1}^2, a_{t-1}^2, R_{t-1}^2, s_t^2)$ into $\mathcal{M}_2$
14 **for** $c \in a_{t-1}^1$ **do**
15 Store $(P_{ic}^{t-1}, a_{t-1}^c, R_{t-1}^c, P_{ic}^t)$ into $\mathcal{M}_c$
16 **end for**
17 **end if**
18 **if** $t \mod K == 0$ **then**
19 Update θ_1 as per Eq. (12)
20 Update $\theta_2^{Q}, \theta_2^{\mu}$ and $\theta_c^{Q}, \theta_c^{\mu}, \ \forall c \in \mathcal{C}$ as per Eq. (14) and (16)
21 Update $\theta_2^{Q'}, \theta_2^{\mu'}$ and $\theta_c^{Q'}, \theta_c^{\mu'}, \ \forall c \in \mathcal{C}$ as per Eq. (17) and (18)
22 Decrease ϵ_1, ϵ_2 and $\epsilon_c, \ \forall c \in \mathcal{C}$
23 **end if**
24 **if** $t \mod K' == 0$ **then**
25 $\theta_1' \leftarrow \theta_1$
26 **end if**
27 **end while**

and calls *Layer1SelectChannels*() to select an action a_t^1; see line 3. The function *Layer1SelectChannels*() selects an action a_t^1 using the function ϵ-greedy(.), where it randomly selects an action with probability ϵ_1. Otherwise, it selects the action with the highest Q-value; see line 1 in Algorithm-2. The value of ϵ_1 is reduced over time until a minimum value of ϵ_{min}. This is to ensure convergence. In addition, the function *Layer1SelectChannels*() also outputs the Layer-2 state s_t^2.

Next, AP i enters Layer-2 with state s_t^2, and calls *Layer2AssignPower*(); see line 4 in Algorithm-1. It first calls ϵ-greedy() to output a vector v, where each element in vector v represents a certain fraction of the maximum transmit power P_{max}. Specifically, the function ϵ-greedy(.) in Layer-2 will use $Simplex(|a_t^1|)$ to sample a random vector v with probability ϵ_2, where $|a_t^1|$ is the number of selected channels. Otherwise, it uses the output of $\mu(s_t^2, \theta_2^{\mu})$ as vector v. Note that we use the *Softmax* function as the activation function for the output layer of DDPG. This ensures the constraints for Layer-2 actions hold, i.e., $0 \leq P_{ic}^t \leq P_{max}$ and $P_{max} = \sum_{c \in C_i^t} P_{ic}^t$. Then, *Layer2AssignPower*() scales the output vector v with the maximum transmit power P_{max} to obtain a Layer-2 action a_t^2, see line 2 in Algorithm-3. Lastly, the function *Layer2AssignPower*() outputs the state s_t^3 for Layer-3.

Algorithm 2 Layer1SelectChannels.

Input: $s_t^1, \theta_1, \epsilon_1$
Output: a_t^1, s_t^2
1 $a_t^1 = \epsilon$-greedy$(s_t^1, \theta_1, \epsilon_1)$
2 $s_t^2 = \{(I_{ic}^t, b_t^c) \mid c \in \mathcal{C}\}$
3 **Return** a_t^1, s_t^2

Algorithm 3 Layer2AssignPower.

Input: $s_t^2, \theta_2^{\mu}, \epsilon_2$
Output: a_t^2, s_t^3
1 $v = \epsilon$-greedy$(s_t^2, \theta_2^{\mu}, \epsilon_2)$
2 $a_t^2 = P_{max} v$
3 $s_t^3 = a_t^2$
4 **Return** a_t^2, s_t^3

In Layer-3, for each channel $c \in a_t^1$, the agent on channel c observes a state, i.e., transmit power $P_{ic}^t \in s_t^3$ and calls *Layer3AdjustCCA*(); see line 5 to 8 in Algorithm-1. The algorithm first calls ϵ-greedy(), where it uniformly samples a value v from the range [0, 1] with probability ϵ_c. Otherwise, it uses the output of $\mu(P_{ic}^t; \theta_c^{\mu})$ as value v. Note that each DDPG agent in Layer-3 uses the *Sigmoid* function as the activation function in the output layer as the CCA threshold on each channel c is a one-dimensional parameter. The value v is then scaled into the range of $[\gamma_{min}, \gamma_{max}]$ to obtain the action a_t^c on channel c; see line 2 in Algorithm-4. Finally, the action of Layer-3 is obtained as $a_t^3 = \{a_t^c \mid c \in a_t^1\}$.

Lastly, the three actions a_t^1, a_t^2 and a_t^3 are executed by AP i to obtain reward R_t^1, R_t^2 and R_t^3. Then, the agent at each layer stores its experience into its memory buffer starting from the second time slot; see line 11 to 16 in Algorithm-1. This is because the state for Layer-2 and Layer-3 depends on the action from their respective upper layer. Therefore, Layer-2 and Layer-3 obtain their respective next state s_{t+1}^2 and s_{t+1}^3 only after Layer-1 and Layer-2 select the action a_{t+1}^1 and a_{t+1}^2 in the following time slot. The stored experiences are then used by AP i to update the neural networks in each layer as shown in line 18 to 21 in Algorithm-1.

VI. EVALUATION

We conducted our simulations using Python 3.7 on a computer with i7-8700 CPU operating at 4.3 GHz and 16 GB RAM.[1] We used TensorFlow 1.14 [45] and Keras 2.2.5 [46]

[1]Note that our agents can also be run on a System-on-Chip (SoC) solution such as Qualcomm QCS605 SoC.

Algorithm 4 Layer3AdjustCCA.

Input: $P_{ic}^t, \theta_c^\mu, \epsilon_c$
Output: a_t^c
$v = \epsilon$-greedy$(P_{ic}^t, \theta_c^\mu, \epsilon_c)$
$a_t^c = v(\gamma_{max} - \gamma_{min}) + \gamma_{min}$
Return a_t^c

TABLE 3. Parameter values.

Environment Parameters	Value(s)
Number of users per AP	4
Number of channels $\|\mathcal{C}\|$	3
Number of neighboring APs $\|\mathcal{N}_i\|$	3
Traffic arrival rate	180 packets per time slot
Available CCA thresholds	-80 to -30 dBm
Total transmit power of each AP [39]	20 dBm
Carrier frequency [39]	5 GHz
Channel bandwidth B [39]	20 MHz
Path loss exponent [40]	3.5
Path loss reference distance d_0 [40]	1 m
Path loss at reference distance $PL(d_0)$ [40]	46.42 dB
Variance for Shadowing effect σ^2	3 dB
Environment noise power density N_b	5×10^{-17} mW/Hz
Initial queue length	8000 packets
Maximum queue length	16000 packets
Packet size L [41]	2304 Bytes
Length of each time slot δ [42]	4.5 ms
RTS/CTS mechanism	Disabled

to build neural networks for our learning agents. These agents ran on an AP, denoted as *i*, that experienced interference from a set of neighboring APs in $\mathcal{N}_i$; each AP is placed 20 m away from AP *i*, acting as the interference source to induce different interference states at AP *i*. There is at least one interfering AP operating on each channel. Each AP had four users that are uniformly placed within 5 m distance. Unless otherwise stated, our simulations used the parameter values listed in Table 3 and 4.

We implemented and compared the following algorithms/rules:

- **DDPG**: AP *i* uses DQN to select a set of channels, and uses *DDPG* for both transmit power distribution and CCA threshold adjustment on each channel.
- **Mixed DDPG and DQN (MixDD)**: AP *i* uses DQN to select channels, and uses *DDPG* for transmit power allocation. It uses *DQN* for CCA threshold adjustment on each channel with eleven discrete CCA threshold values, ranging from -80 to -30 dBm.
- **DQN**: AP *i* uses DQN to select channels, and uses *DQN* for both transmit power distribution and CCA threshold adjustment on each channel. We discretized the AP's transmit power into eight levels, ranging from zero to 100 Watts. The CCA threshold is discretized into 11 levels, ranging from -80 to -30 dBm.
- **All Channels Bonded (ACB)**: AP *i* will always use all channels for transmissions. The transmit power on each channel is evenly distributed. AP *i* uses a CCA threshold of -82 dBm.[2]
- **Random**: AP *i* randomly selects a set of channels for transmissions. The transmit power on each selected channel is distributed evenly. The CCA threshold for each channel is set to -82 dBm.
- **Primary Channel Only (PCO)**: AP *i* randomly selects a channel as its primary channel, and only uses the primary channel for transmissions with the maximum transmit power. The CCA threshold for the primary channel is set to -82 dBm.

TABLE 4. Parameter values used by learning agents [12], [43], [44].

Parameters	Value(s)
Number of hidden layers	2
Number of neurons in each layer	32
Activation function for hidden layers	ReLU
Learning rate α	10^{-3}
Optimizer	Adam
Reward discount factor γ	0.95
Weight of data rate η_1	0.9
Weight of queue length η_2	0.01
Initial exploration rate ϵ	1
Final exploration rate ϵ_{min}	0.1
Decay factor ϵ_d	0.9995
Memory size (samples)	5000
Batch size for training (samples)	32
Interval K to update θ	Every five time slots
Interval K' to replace DQN θ'	Every 500 time slots
DDPG target network update rate τ	0.005
Number of learning time slots N_L	40000

Note that *MixDD* and *DQN* are two variations of *DDPG*. The motivation for studying them is to investigate the case where *DDPG* uses a different reinforcement learning algorithm in each layer.

Our simulations used episodes that comprised of 500 time slots. For each episode, we collected the following metrics:

- **Average number of transmitted packets**: This is the average number of packets transmitted per time slot by AP *i* in an episode.

[2]This value is set as per the IEEE WiFi standard [41].

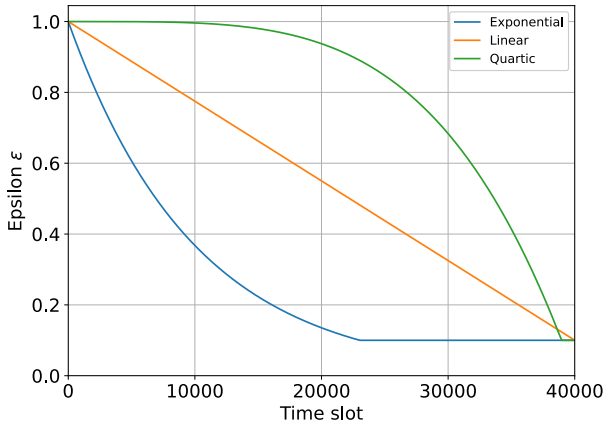


FIGURE 4. Elapsed time versus value of Epsilon.

- **Average queue length of an AP**: This is the average queue length of AP i collected at the end of each time slot in each episode.

Our simulations had *three* stages: *Training*, *Test* and *Supplementary*. In the Training stage, AP i had three available channels, and on each channel, there was an interfering AP located 20 m away. AP i always have packets to transmit. The packet size is fixed to 2304 Bytes [41]. In this stage, we also conducted simulations to study the impact of ϵ decay rules and values of η_1 and η_2. To train our agents, for the first 5000 time slots, we programmed agents to randomly select actions to ensure they collected sufficient data. Next, we trained our agents for 40000 time slots. After that, we set the value of ϵ to zero and ran the simulation for another 5000 time slots to analyze their convergence performance.

In the Test stage, we used the same network model as the Training stage, and studied two traffic models, number of interfering APs and channel gain variance. We used a Poisson traffic model to control the number of packets that arrived at AP i in each time slot; its arrival rate ranged from 30 to 240 packets per time slot. We have also constructed a traffic model using the trace data in [47]. After that, we studied the impact of interfering APs on each channel, which ranged from one to six on each channel, meaning the number of interfering APs increased from three to 18. Next, we evaluated the impact of channel gain variance or the shadowing term σ^2, which ranged from zero to 80 dB.

Next, in the Supplementary stage, we outline our performance study of DDPG, MixDD and DQN when there are different numbers of channels, which ranged from two to eight; each channel had one interfering AP. Lastly, we investigated the topology and channel model provided by the IEEE 802.11ax task group [48].

A. TRAINING STAGE

1) ϵ DECAY RULES

We evaluated three ϵ decay rules. The value of ϵ at each time slot t was calculated as follows:

- **Exponential**: $\epsilon^t = \max(\epsilon_d^{t/K}, \epsilon_{min})$.

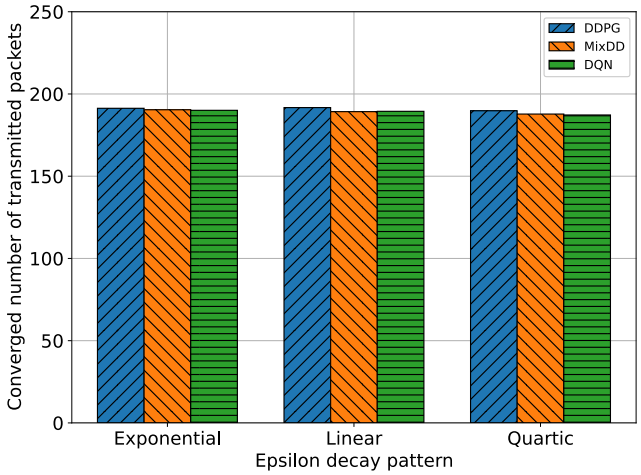


FIGURE 5. Converged throughput versus Epsilon decay patterns.

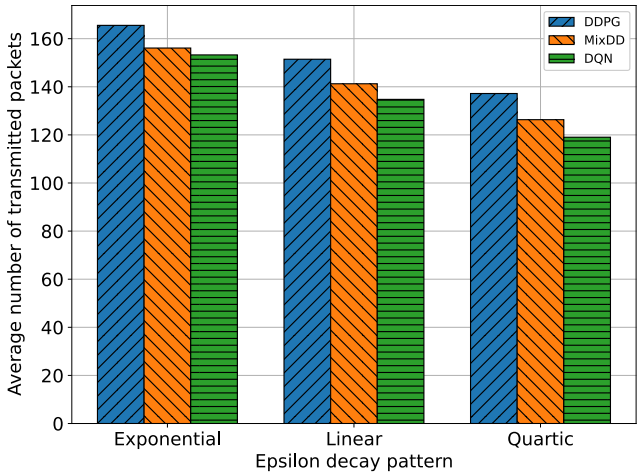


FIGURE 6. Average throughput versus Epsilon decay patterns.

- **Linear**: $\epsilon^t = \max(1 - \frac{(1-\epsilon_{min})t}{N_L/K}, \epsilon_{min})$.
- **Quartic**: $\epsilon^t = \max(1 - (\frac{t}{N_L/K})^4, \epsilon_{min})$.

In our discussion to follow, the term K and N_L, refer respectively to the learning frequency, and number of time slots for learning. Fig. 4 shows the evolution of ϵ over time. Our simulations showed that for the *Exponential* rule, the value of ϵ reduced at the fastest rate before 15000 time slots, and then it reduced at a slower rate than the *Quartic* and *Linear* rule. By contrast, the value of ϵ for the *Quartic* rule decreased at the lowest rate at the beginning and started to decrease faster after 10000 time slots. Note that the value of ϵ will not decrease below the minimum value of ϵ_{min}.

Referring to Fig. 5, DDPG, MixDD and DQN converged to around 190 packets per time slot. In addition, Fig. 5 shows that different ϵ decay rules have no significant impact on the converged throughput for all tested algorithms. We recorded the largest difference of 4.5 packets per time slot between DDPG with the *Linear* rule and DQN with the *Quartic* rule, which only differed by 2.39%. Fig. 6 shows the average throughput for different ϵ decay rules. The *Exponential* rule

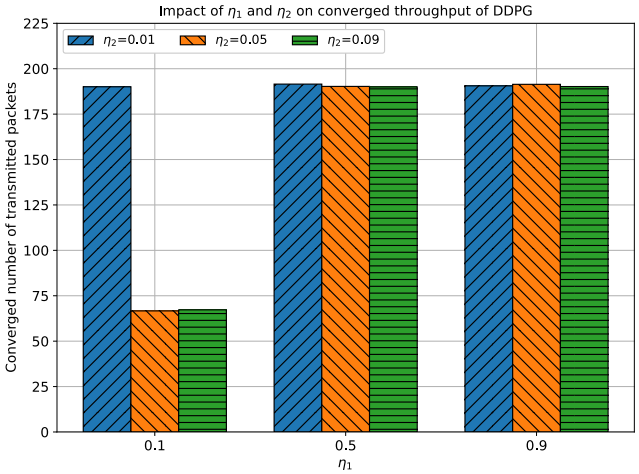


FIGURE 7. **Converged throughput versus values of η_1 and η_2.**

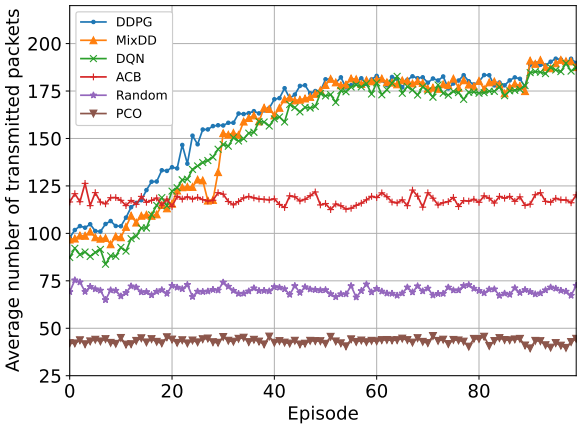


FIGURE 8. **Elapsed time versus the number of transmitted packets.**

had the highest average throughput for all three learning algorithms. The average number of transmitted packets per time slots for DDPG, MixDD and DQN was 165.5, 156.1 and 153.2, respectively, meaning the *Exponential* rule outperformed the *Linear* rule by 11.18% and the *Quartic* rule by 24.32%, on average. This is because when ϵ reduced to a low value, an agent will exploit its learned policy with a high probability. The *Exponential* rule reduced ϵ to the minimum value ϵ_{min} at the fastest rate. In addition, as per Fig. 5, all three ϵ decay rules converged to 190 packets per time slot. Consequently, the *Exponential* rule had the highest average throughput among all ϵ decay rules. Hence, in all subsequent simulations, we will use the *Exponential* rule as the ϵ decay rule when training agents.

2) IMPACT OF η_1 AND η_2

We have also evaluated the impact of different values for η_1 and η_2. We trained DDPG with an η_1 and η_2 value drawn respectively from the range $[0.1, 0.5, 0.9]$, and $[0.01, 0.05, 0.09]$.

Fig. 7 shows the converged throughput for different combination of η_1 and η_2 values. We see that when η_1 equals 0.1, the converged throughput for $\eta_2 = 0.05$ and $\eta_2 = 0.09$ is respectively 66.7 and 67.2 packets per time slot. DDPG achieved an average number of transmitted packets that exceeded 190 packets per time slot for all other combinations of η_1 and η_2. Recall that η_1 and η_2 weigh the importance of data rate and queue length, respectively. The results in Fig. 7 suggest that the value of η_1 should be sufficiently larger than η_2 to balance the impact of data rate and queue length on the reward received by an AP. Otherwise, a minor change in queue length could have a significant impact on the AP's reward, which may potentially cause an agent to learn an incorrect action or policy. Hence, in all subsequent simulations, we set the value of η_1 to 0.9 and η_2 to 0.01.

3) CONVERGENCE

Fig. 8 shows that the average number of packets transmitted by DDPG, MixDD and DQN increased over time. This average value for DDPG, MixDD and DQN respectively increased from 102.9, 97.8 and 88.8, and converged to 190.2, 189.6 and 186.0 packets per time slot. This is because they were able to learn the optimal policies for channel selection, transmit power distribution and CCA threshold adjustment over time. Fig. 8 also shows that the performance for DDPG is better than the other two learning algorithms. For example, the number of transmitted packets per time slot for DDPG is 4.43% and 6.84% higher than that of MixDD and DQN on average. This is because DDPG used a continuous action space for its transmit power distribution and CCA threshold. By contrast, MixDD employed discrete CCA thresholds on each channel whereas DQN learned over both discrete power distribution and CCA threshold action space. Thus, MixDD and DQN failed to learn the optimal action when their action space is not discretized. Therefore, DDPG outperformed the other two learning algorithms. We also see that ACB, Random and PCO had the same average number of transmitted packets over time; i.e., 118.6, 70.6 and 42.8 packets per time slot, respectively. This is because these three algorithms had no learning mechanism to optimize channel usage, transmit power and CCA threshold.

B. TEST STAGE

1) POISSON TRAFFIC

To study the impact of Poisson traffic, we assumed three channels. We varied the traffic arrival rate from 30 to 240 packets per time slot. Simulations ran for 5000 time slots for each traffic arrival rate. Fig. 9 and 10 show the impact of arrival rates. From Fig. 9, DDPG, MixDD and DQN had an increasing throughput trend. They were able to transmit 31.56 packets per time slot when the traffic arrival rate was set to 30 packets per time slot. This number then increased to 190.8 for DDPG, 188.8 for MixDD and 186.6 for DQN when the traffic arrival rate was set to 240 packet per time slot. This is because these three learning algorithms were able to transmit more than 186 packets per time slot after training as shown in Fig. 8. When the traffic arrival rate was

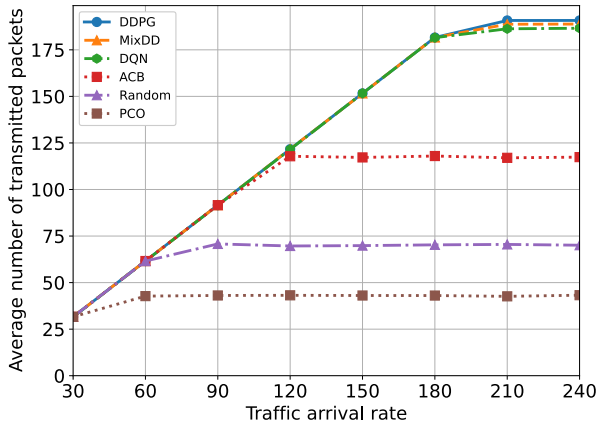


FIGURE 9. Impact of traffic arrival rate on throughput.

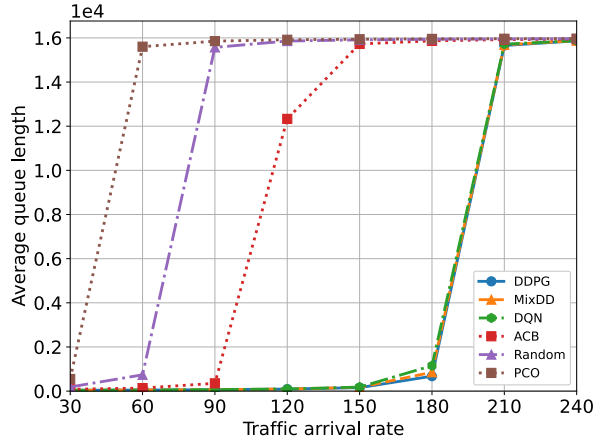


FIGURE 10. Impact of traffic arrival rate on average queue length.

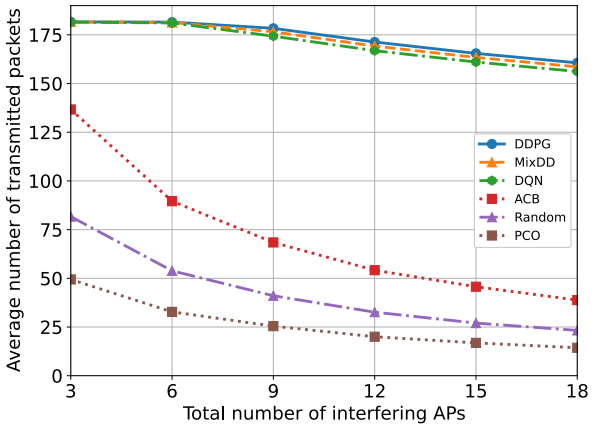


FIGURE 11. Impact of number of Interfering APs on throughput.

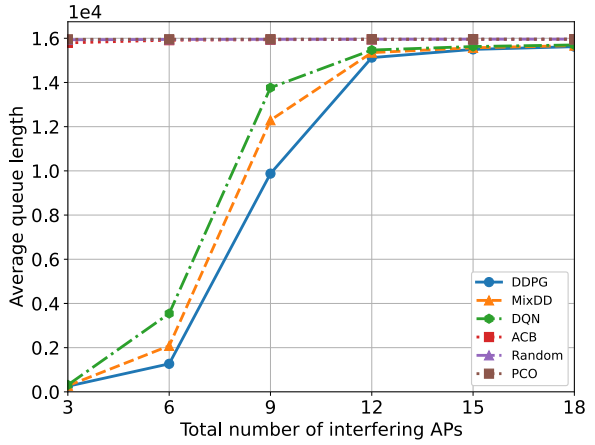


FIGURE 12. Impact of number of Interfering APs on average queue length.

lower than 180 packets per time slot, DDPG, MixDD and DQN were able to empty the queue of AP *i*. As a result, its throughput was limited by a low traffic arrival rate. Fig. 10 shows that the average queue length of DDPG, MixDD and DQN is lower than 1100 packets when AP *i* experienced a traffic arrival rate no larger than 180 packets per time slot. However, when the traffic arrival rate exceeded 180 packets per time slots, DDPG, MixDD and DQN did not have sufficient throughput to deliver all arriving packets, which increased the queue length of AP *i*. From Fig. 10, the average queue length of these three learning algorithms increased significantly from 1500 to 15968 packets when the traffic arrival rate increased from 180 to 210 packets per time slot. We observed similar trends for ACB, Random and PCO, where they had an average throughput of 117.2, 71.8 and 42.4 packets per time slot as shown in Fig. 8. This means they were only able to reduce the queue length of AP *i* when its traffic arrival rate is lower than the average throughput.

2) INTERFERING APs

Here, APs have Poisson traffic with an arrival rate of 180 packets per time slot. There were three channels, and one to six interfering APs on each channel. We uniformly placed all interfering APs within the range of 40 m around AP *i*. For each number of interfering APs, we ran the simulation ten times, with 5000 time slots in each run.

Fig. 11 shows that the throughput of AP *i* decreased when there are more interfering APs. DDPG, MixDD and DQN were able to transmit 181 packets per time slot when there were three interfering APs. This number decreased to 160.6 for DDPG, 158.6 for MixDD and 156.2 for DQN when the number of interfering APs increased to 18. As a comparison, the average number of transmitted packets per time slot for ACB, Random and PCO decreased from 136.7, 81.7 and 49.4 to 38.8, 23.3 and 14.3, respectively. This is because the level of interference on each channel increased as the number of interfering APs increased, which led to throughput degradation for all algorithms/rules. However, DDPG, MixDD and DQN continued to have better performance against higher interference levels as compared to ACB, Random and PCO. The throughput of DDPG, MixDD and DQN reduced by 12.74% on average as the number of interfering APs increased from three to 18. In contrast, the performance

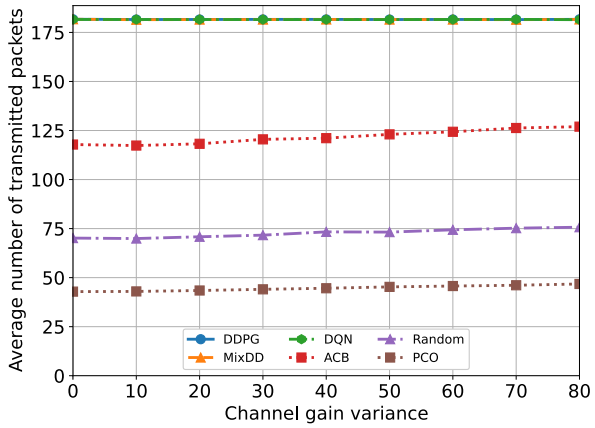


FIGURE 13. Impact of channel gain variance on throughput.

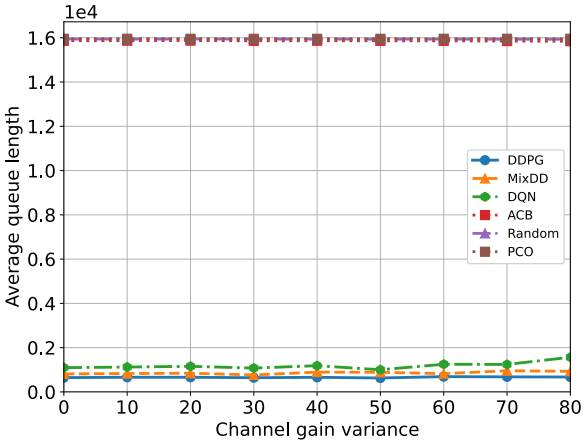


FIGURE 14. Impact of channel gain variance on the average queue length of APs.

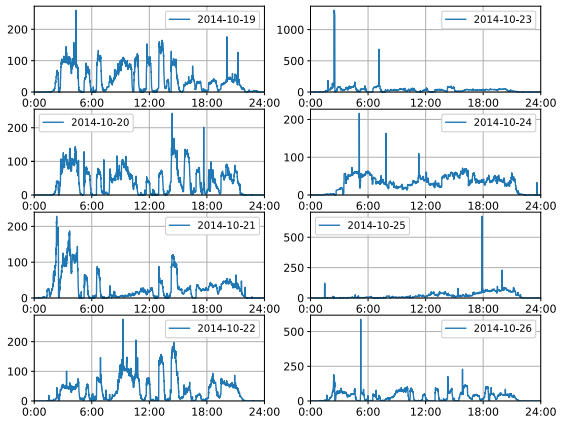


FIGURE 15. Arriving traffic for AP *i* throughout eight days. The x-axis represents the time across a day, and y-axis represents the number of packets arriving in each time slot, respectively.

of ACB, Random and PCO dropped by 71.37% on average under the same circumstance. From Fig. 11, DDPG, MixDD and DQN always had the highest throughput. In particular, these three learning algorithms achieved 137.72%, 296.67% and 548.5% higher throughput than ACB, Random and PCO on average, respectively. This is because these three learning algorithms were able to learn the optimal channel selection, transmit power distribution and CCA threshold adjustment for varying interference levels. From Fig. 11, DDPG had a higher throughput as compared to MixDD by 1.21% and DQN by 2.65% for six interfering APs scenario. The difference in throughput led to different average queue lengths. Referring to Fig. 12, DDPG had a queue length that was 5.88% and 10.51% shorter than MixDD and DQN. Hence, using a continuous action space for transmit power and CCA threshold led to better performance than a discrete action space. In contrast, the average queue length for ACB, Random and PCO was always around 16000 packets. These three rules were not able to transmit more than 180 packets per time slot. Therefore, they were not able to reduce the queue length of AP *i*, resulting in queue overflow.

3) CHANNEL GAIN VARIANCE

To study the impact of channel gain variance, we have used the following settings: Poisson traffic with an arrival rate of 180 packets per time slot, and three channels. The channel gain variance σ^2 was increased from zero to 80 dB. Referring to Fig. 13 and 14, the performance of DDPG, MixDD and DQN was not affected by the changing channel gain variance. Fig. 13 shows that DDPG, MixDD and DQN were able to transmit 181 packets per time slots for all channel gain variance values. The average queue length of these three algorithms was less than 2000 packets. This is because DDPG, MixDD and DQN were able to learn the optimal transmit power and CCA threshold for each channel that resulted in a high throughput against various levels of interference. In contrast, the throughput of ACB, Random and PCO increased with higher channel gain variance. The average number of transmitted packets per time slot for ACB, Random and PCO increased from 117.8 to 127.0, 70.2 to 75.7 and 42.8 to 46.8, respectively. Their throughput improved by 8.25% on average as the channel gain variance increased from zero to 80. This is because as the variance increased, the corresponding Cumulative Distribution Function (CDF) changed, which increased the probability that the interference on each channel to be less than the CCA threshold used by AP *i*, i.e., -82 dBm. Therefore, for high channel gain variance, ACB, Random and PCO had more opportunities to transmit than when channel gain variance was low. Consequently, the average number of transmitted packets increased. However, the increased in throughput was lower than the traffic arrival rate. Hence, ACB, Random and PCO were not able to reduce the queue length of AP *i*, and suffered from queue overflow. Referring to Fig. 14, the average queue length for ACB, Random and PCO stayed at the maximum queue length of 16000 packets for all channel gain variances.

4) TRACE-BASED STUDY

The next simulation evaluated the performance of all algorithms/rules using the traffic trace file provided in [47]. We extracted eight days of traffic, from 19 October

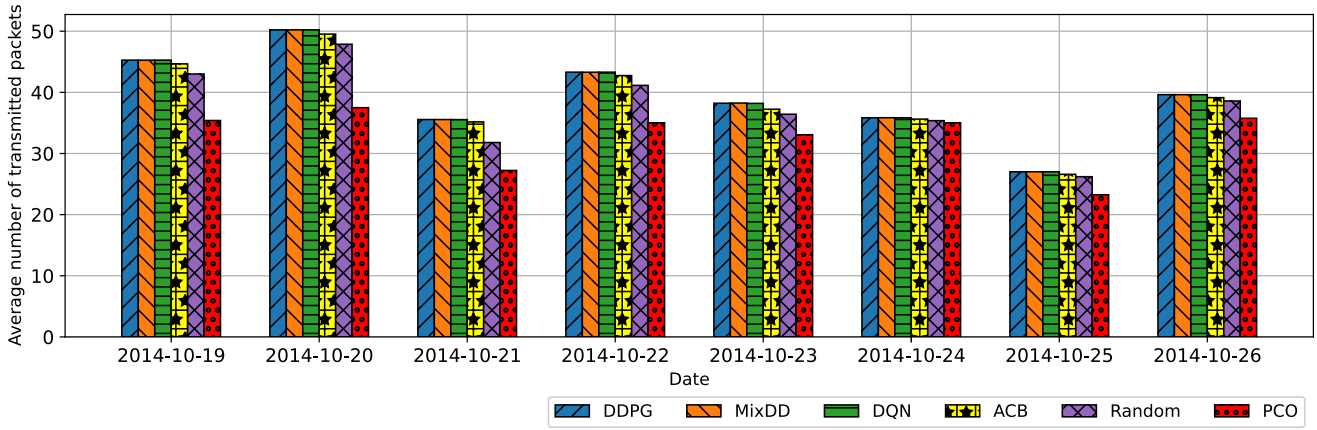


FIGURE 16. **Average throughput with difference trace data.**

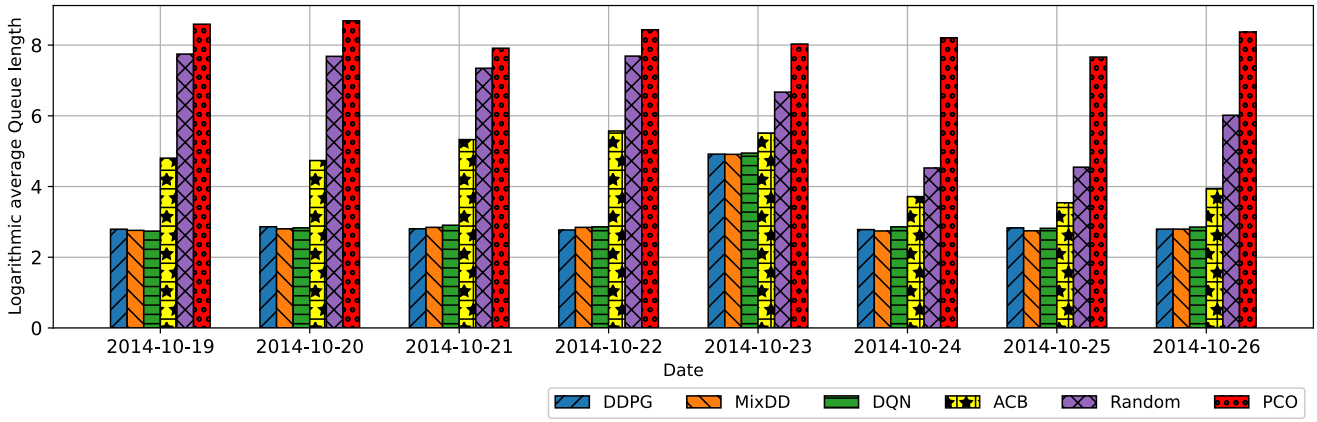


FIGURE 17. **Average queue length with difference trace data. Note that we have taken the *logarithmic* value of the original queue length (in packets).**

2014 to 26 October 2014, see Fig. 15, and used it as the arriving traffic for AP *i*. Fig 16 and 17 show the average number of transmitted packets and queue length for each dated traffic trace.

From Fig 16 and 17, DDPG, MixDD and DQN had the highest throughput and lowest queue length. In Fig 16, the average number of transmitted packets per time slot for DDPG, MixDD and DQN was around 39.4. As a comparison, ACB, Random and PCO were able to transmit 38.8, 37.5 and 32.8 packets per time slot. The average throughput of DDPG, MixDD and DQN was only 1.4%, 4.87% and 20.13% higher than ACB, Random and PCO, respectively. The reason was because the arriving traffic rate was low throughout a day. From Fig. 15, the average number of arriving packets in each time slot was around 30.6 across the eight days. Therefore, DDPG, MixDD and DQN were able to empty the queue of APs quickly. This can also be seen from Fig. 17, where DDPG, MixDD and DQN had the smallest average queue length of 3.05 packets. As a result, DDPG, MixDD and DQN did not have a large number of packets to transmit in each time slot, which resulted in a low average number of transmitted packets per time slot. In contrast, ACB, Random and PCO had an average queue length of 4.64, 6.52 and 8.24, meaning the average queue length of DDPG, MixDD and DQN was 32.94%, 50.99% and 62.52% shorter than ACB, Random and PCO.

C. SUPPLEMENTARY STAGE

1) NUMBER OF CHANNELS

We now present the simulation that studied different number of channels. Each channel had one interfering AP placed 20 m away from AP *i*. As the number of channels differs, agents used a different action space. Therefore, for each channel number, we re-built our learning agents, and trained them until convergence. Fig. 18 shows the converged throughput for different number of channels. The performance of DDPG, MixDD and DQN had no difference when the number of channels was no larger than four. The average number of transmitted packets per time slot for DDPG, MixDD and DQN increased from 130 to 245 when the number of channels increased from two to four. This means all three learning agents were able to learn the optimal policy when the number of channels was low. However, as the number of channels

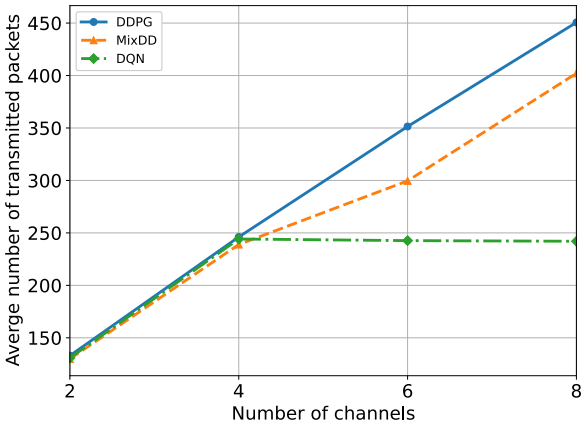


FIGURE 18. Converged throughput verses number of channels.

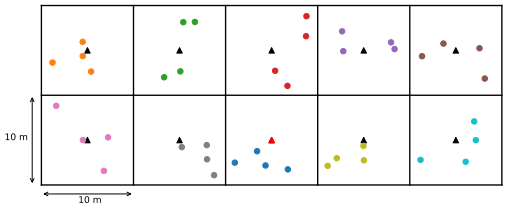


FIGURE 19. The topology of IEEE simulation scenario for apartment [48], where each apartment has a dimension of 10 m × 10 m. Each AP is placed in the center of an apartment. The target AP *i* (red triangle) is located at the center bottom apartment and all other APs (black triangles) act as the interference sources. The penetration loss of each wall is set to 5 dB [49].

increased, DDPG achieved the highest throughput. The average number of transmitted packets per time slot for DDPG increased from 351 to 451 when the number of channels increased from six to eight, which was 14.67% and 65.51% higher than MixDD and DQN on average, respectively. This means DDPG was able to learn the optimal policy in scenarios with different number of channels. In contrast, the throughput of DQN showed no improvement when the simulation used four channels. The converged throughput remained around 243 packets per time slot as the number of channels increased from four to eight. This is because the action space for DQN increased significantly with more channels. For example, the number of actions for the action space related to transmit power over six and eight channels was 1287 and 6435. As a result, agents were not able to explore and learn each action efficiently during training. Therefore, agents were not able to learn the optimal policy, which resulted in poor performance.

2) IEEE SCENARIOS

We have also conducted simulations using the IEEE scenario and channel model proposed in [48] and [49]. Referring to Fig. 19, we simulated an apartment with ten cells; each cell had a dimension of 10 m × 10 m. We placed an AP at the center of each cell, and selected the AP in the center bottom cell as AP *i* that ran our learning agents. Each AP had four

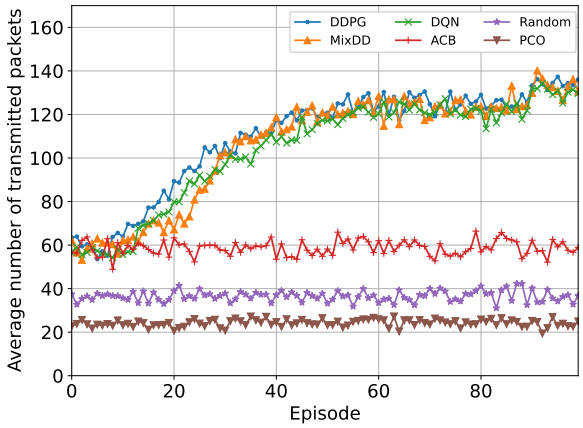


FIGURE 20. Elapsed time versus average number of transmitted packets with IEEE simulation scenario.

associated users that are uniformly placed within its cell. We adopted the indoor channel model provided in [49], where the wall penetration loss was set to 5 dB.

Fig. 20 shows the average number of transmitted packets for different algorithms/rules. We see that DDPG, MixDD and DQN were able to achieve the highest throughput over time. The average number of transmitted packets per time slot for DDPG, MixDD and DQN increased from 61.2, 59.3 and 56.8 to 135.0, 133.0 and 130.6, respectively. These three learning algorithms were able to learn the optimal policy for the stated IEEE scenario and channel model. DDPG achieved the highest throughput among all three learning algorithms, where its average number of transmitted packets per time slot was 4.01% and 5.83% higher than that of MixDD and DQN. In contrast, the throughput for ACB, Random and PCO remained the same over time. ACB, Random and PCO achieved an average number of transmitted packets of 58.5, 36.6 and 23.5 per time slot.

VII. CONCLUSION

This paper has outlined and studied a novel three-tier learning approach that aims to improve multi-channel utilization and minimize the queue length of an AP operating in a WiFi network. Specifically, the AP uses our approach to learn a policy that governs when and how it uses one or more channels, allocate its transmit power and set the CCA threshold for each selected channel given varying environmental conditions. Advantageously, the proposed approach requires an AP to use only local information, such as its queue length and observed interference level. The simulation results showed that the proposed learning approach was able to learn the optimal policy, and achieved the best performance under multiple scenarios. Numerical results showed that the proposed three-tier learning approach was able to reduce the average queue length of an AP by up to 62.52% when compared to an AP that used a fixed strategy over realistic traffic trace data. An interesting future work is to consider time-varying number of users with different quality of service requirements such as data rate or

transmission frequency. In this respect, the agent will have to incorporate into its state the number of users, and their requirements. Its goal is then to learn a policy that ensures users meet their respective requirements. Another possibility is to leverage generative artificial intelligence or diffusion models to speed up training or to improve an AP's policy. Specifically, an AP can be first trained offline using data generated from such models to obtain a preliminary policy for a given environment. After that, the AP/agent uses actual system states to refine its policy.

REFERENCES

[1] E. Khorov, A. Kiryanov, A. Lyakhov, and G. Bianchi, "A tutorial on IEEE 802.11ax high efficiency WLANs," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 197–216, 1st Quart., 2019.

[2] S. Barrachina-Muñoz, B. Bellalta, and E. W. Knightly, "Wi-Fi channel bonding: An all-channel system and experimental study from urban hotspots to a sold-out stadium," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2101–2114, Oct. 2021.

[3] R. Gummadi, D. Wetherall, B. Greenstein, and S. Seshan, "Understanding and mitigating the impact of RF interference on 802.11 networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 385–396, Oct. 2007.

[4] K. K. Sreedhar, A. Aminlou, M. M. Hannuksela, and M. Gabbouj, "Viewport-adaptive encoding and streaming of 360-degree video for virtual reality applications," in *Proc. IEEE Int. Symp. Multimedia (ISM)*, San Jose, CA, USA, Dec. 2016, pp. 583–586.

[5] B. Bellalta, "IEEE 802.11ax: High-efficiency WLANS," *IEEE Wireless Commun.*, vol. 23, no. 1, pp. 38–46, Feb. 2016.

[6] B. Alawieh, Y. Zhang, C. Assi, and H. Mouftah, "Improving spatial reuse in multihop wireless networks—A survey," *IEEE Commun. Surveys Tuts.*, vol. 11, no. 3, pp. 71–91, Aug. 2009.

[7] C. Thorpe and L. Murphy, "A survey of adaptive carrier sensing mechanisms for IEEE 802.11 wireless networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1266–1293, 3rd Quart., 2014.

[8] L. Deek, E. Garcia-Villegas, E. Belding, S.-J. Lee, and K. Almeroth, "The impact of channel bonding on 802.11n network management," in *Proc. 7th Conf. Emerg. Netw. Experiments Technol.*, Japan, Dec. 2011, pp. 1–12.

[9] S. Barrachina-Muñoz, F. Wilhelmi, and B. Bellalta, "To overlap or not to overlap: Enabling channel bonding in high-density WLANs," *Comput. Netw.*, vol. 152, pp. 40–53, Apr. 2019.

[10] S. Jang, K. G. Shin, and S. Bahk, "Post-CCA and reinforcement learning based bandwidth adaptation in 802.11ac networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 2, pp. 419–432, Feb. 2018.

[11] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: Wiley, 1994.

[12] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[13] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *Proc. ICLR*, San Juan, Puerto Rico, May 2016, pp. 1–14.

[14] A. Nabil, M. J. Abdel-Rahman, and A. B. MacKenzie, "Adaptive channel bonding in wireless LANs under demand uncertainty," in *Proc. IEEE PIMRC Conf.*, Montreal, QC, Canada, Oct. 2017, pp. 1–7.

[15] T. Moscibroda, R. Chandra, Y. Wu, S. Sengupta, P. Bahl, and Y. Yuan, "Load-aware spectrum distribution in wireless LANs," in *Proc. IEEE Int. Conf. Netw. Protocols*, Orlando, FL, USA, Oct. 2008, pp. 137–146.

[16] S. Lee, T. Kim, S. Lee, K. Kim, Y. H. Kim, and N. Golmie, "Dynamic channel bonding algorithm for densely deployed 802.11ac networks," *IEEE Trans. Commun.*, vol. 67, no. 12, pp. 8517–8531, Dec. 2019.

[17] Y. Chen, D. Wu, T. Sung, and K. Shih, "DBS: A dynamic bandwidth selection MAC protocol for channel bonding in IEEE 802.11ac WLANs," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Barcelona, Spain, Apr. 2018, pp. 1–6.

[18] X. Wang, P. Huang, J. Xie, and M. Li, "OFDMA-based channel-width adaptation in wireless mesh networks," *IEEE Trans. Veh. Technol.*, vol. 63, no. 8, pp. 4039–4052, Oct. 2014.

[19] S. Rayanchu, V. Shrivastava, S. Banerjee, and R. Chandra, "FLUID: Improving throughputs in enterprise wireless LANs through flexible channelization," *IEEE Trans. Mobile Comput.*, vol. 11, no. 9, pp. 1455–1469, Sep. 2012.

[20] T. Song, T. Kim, W. Kim, and S. Pack, "Channel bonding algorithm for densely deployed wireless LAN," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Kota Kinabalu, Malaysia, Jan. 2016, pp. 395–397.

[21] T. Song, T. Kim, W. Kim, and S. Pack, "Adaptive and distributed radio resource allocation in densely deployed wireless LANs: A game-theoretic approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4466–4475, May 2018.

[22] C. Kai, Y. Liang, T. Huang, and X. Chen, "To bond or not to bond: An optimal channel allocation algorithm for flexible dynamic channel bonding in WLANs," in *Proc. IEEE 86th Veh. Technol. Conf. (VTC-Fall)*, Toronto, ON, Canada, Sep. 2017, pp. 1–6.

[23] M. Han, S. Khairy, L. X. Cai, Y. Cheng, and F. Hou, "Capacity analysis of opportunistic channel bonding over multi-channel WLANs under unsaturated traffic," *IEEE Trans. Commun.*, vol. 68, no. 3, pp. 1552–1566, Mar. 2020.

[24] R. Karmakar, S. Chattopadhyay, and S. Chakraborty, "SmartBond: A deep probabilistic machinery for smart channel bonding in IEEE 802.11ac," in *Proc. IEEE Conf. Comput. Commun.*, Toronto, ON, Canada, Jul. 2020, pp. 2599–2608.

[25] H. Qi, H. Huang, Z. Hu, X. Wen, and Z. Lu, "On-demand channel bonding in heterogeneous WLANs: A multi-agent deep reinforcement learning approach," *Sensors*, vol. 20, no. 10, pp. 1–16, May 2020.

[26] Y. Luo and K. Chin, "Learning to bond in dense WLANs with random traffic demands," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 11868–11879, Oct. 2020.

[27] M. Han, Z. Chen, L. X. Cai, T. H. Luan, and F. Hou, "A deep reinforcement learning based approach for channel aggregation in IEEE 802.11 ax," in *Proc. IEEE Global Commun. Conf.*, Dec. 2020, pp. 1–6.

[28] R. Karmakar and G. Kaddoum, "IBAC: An intelligent dynamic bandwidth channel access avoiding outside warning range problem," *IEEE Trans. Mobile Comput.*, vol. 22, no. 6, pp. 3350–3364, Jun. 2023.

[29] Y. Luo and K. Chin, "An energy efficient channel bonding and transmit power control approach for WiFi networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 8, pp. 8251–8263, Aug. 2021.

[30] M. Bennis and D. Niyato, "A Q-learning based approach to interference avoidance in self-organized femtocell networks," in *Proc. IEEE Globecom Workshops*, Miami, FL, USA, Dec. 2010, pp. 706–710.

[31] W. Wang, F. Zhang, and Q. Zhang, "Managing channel bonding with clear channel assessment in 802.11 networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kuala Lumpur, Malaysia, May 2016, pp. 1–6.

[32] L. Lanante and S. Roy, "Analysis and optimization of channel bonding in dense IEEE 802.11 WLANs," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 2150–2160, Mar. 2021.

[33] F. Wilhelmi, S. Barrachina-Muñoz, B. Bellalta, C. Cano, A. Jonsson, and G. Neu, "Potential and pitfalls of multi-armed bandits for decentralized spatial reuse in WLANs," *J. Netw. Comput. Appl.*, vol. 127, pp. 26–42, Feb. 2019.

[34] Y. Huang and K. Chin, "A deep Q-network approach to optimize spatial reuse in WiFi networks," *IEEE Trans. Veh. Technol.*, vol. 71, no. 6, pp. 6636–6646, Jun. 2022.

[35] T. S. Rappaport, *Wireless Communications: Principles and Practice*. Upper Saddle River, NJ, USA: Prentice-Hall, Sep. 2002.

[36] N. A. Smith and R. W. Tromble, "Sampling uniformly from the unit simplex," Johns Hopkins Univ., Baltimore, MD, USA, Tech. Rep., vol. 29, 2004.

[37] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn.*, Beijing, China, Jan. 2014, pp. 387–395.

[38] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor, "Learn what not to learn: Action elimination with deep reinforcement learning," in *Proc. NeurIPS*, Montréal, QC, Canada, Dec. 2018, pp. 3562–3573.

[39] B. Bellalta, L. Bononi, R. Bruno, and A. Kassler, "Next generation IEEE 802.11 wireless local area networks: Current status, future directions and open challenges," *Comp. Commun.*, vol. 75, pp. 1–25, Feb. 2016.

[40] M. Viswanathan. (Sep. 2013). *Log Distance Path Loss or Log Normal Shadowing Model*. Accessed: Apr. 12, 2022. [Online]. Available: https://www.gaussianwaves.com/2013/09/log-distance-path-loss-or-log-normal-shadowing-model/

[41] *IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless Lan Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Standard 802.11-2016, Dec. 2016, p. 3534.

[42] B. Yin, K. Yamamoto, T. Nishio, M. Morikura, and H. Abeysekera, "Learning-based spatial reuse for WLANs with early identification of interfering transmitters," *IEEE Trans. Cognit. Commun. Netw.*, vol. 6, no. 1, pp. 151–164, Mar. 2020.

[43] Y. Li, W. Zhang, C. Wang, J. Sun, and Y. Liu, "Deep reinforcement learning for dynamic spectrum sensing and aggregation in multi-channel wireless networks," *IEEE Trans. Cognit. Commun. Netw.*, vol. 6, no. 2, pp. 464–475, Jun. 2020.

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, May 2015, pp. 1–15.

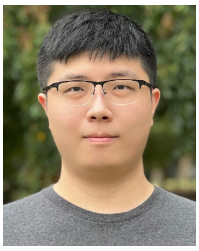
[45] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.

[46] F. Chollet et al., (2015). *Keras*. Accessed: Feb. 2, 2019. [Online]. Available: https://keras.io/

[47] J. Liu, B. Krishnamachari, S. Zhou, and Z. Niu, "DeepNap: Data-driven base station sleeping operations through deep reinforcement learning," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4273–4282, Dec. 2018.

[48] S. Merlin et al., (Jul. 2015). *TGax Simulation Scenarios Document IEEE 802.11-14/0980r16*. Accessed: Apr. 12, 2022. [Online]. Available: https://mentor.ieee.org/802.11/dcn/14/11-14-0980-16-00ax-simulation-scenarios.docx

[49] J. Liu et al., (Sep. 2014). *IEEE 802.11ax Channel Model Document*. Accessed: Apr. 12, 2022. [Online]. Available: https://mentor.ieee.org/802.11/dcn/14/11-14-0882-04-00ax-tgax-channel-model-document.docx



YIWEI HUANG received the bachelor's degree (Hons.) in telecommunications engineering from the University of Wollongong, Australia, in 2016, and the master's degree in data science from The University of Sydney, Australia, in 2017. He is currently pursuing the Ph.D. degree with the University of Wollongong. His current research interests include performance optimization in wireless networks and reinforcement learning.

KWAN-WU CHIN received the B.Sc. degree (Hons.) and the Ph.D. degree (with the vice-chancellor commendation) from Curtin University, Australia, in 1997 and 2000, respectively. From 2000 to 2003, he was a Senior Research Engineer with Motorola. In 2004, he joined the University of Wollongong, Australia, as a Senior Lecturer, and was promoted to an Associate Professor in 2011, where he is currently an Associate Professor. To date, he holds four United States (U.S.) patents, and has published more than 190 conference and journal papers. His current research interests include medium access control protocols for wireless networks, and resource allocation algorithms/policies for communications networks.