Accelerated Structure-Aware Reinforcement Learning for Delay-Sensitive Energy Harvesting Wireless Sensors

Nikhilesh Sharma, Nicholas Mastronarde, *Senior Member, IEEE*, and Jacob Chakareski, *Senior Member, IEEE*

***Abstract*—We consider a time-slotted energy-harvesting wireless sensor transmitting delay-sensitive data over a fading channel. The sensor injects captured data packets into its transmission queue and relies on ambient energy harvested from the environment to transmit them. We aim to find the optimal scheduling policy that decides how many packets to transmit in each time slot to minimize the expected queuing delay. No prior knowledge of the stochastic processes that govern the channel, captured data, and harvested energy dynamics is assumed, thereby necessitating online learning to optimize the scheduling policy. We formulate this problem as a Markov decision process (MDP) with state-space spanning the sensor's buffer, battery, and channel states, and show that its optimal value function is non-decreasing and has increasing differences, in the buffer state, and that it is non-increasing and has increasing differences, in the battery state. We exploit this value function structure knowledge to formulate a novel accelerated reinforcement learning (RL) algorithm based on value function approximation that can solve the scheduling problem online with controlled approximation error, while inducing limited computational and memory complexity. We rigorously capture the tradeoff between approximation accuracy and computational/memory complexity savings associated with our approach. Our simulations demonstrate that the proposed algorithm closely approximates the optimal offline solution, which requires complete knowledge of the system state dynamics. Simultaneously, our approach achieves competitive performance relative to a state-of-the-art RL algorithm, at orders of magnitude lower complexity. Moreover, considerable performance gains are demonstrated over the widely popular Q-learning RL technique.**

***Index Terms*—Energy harvesting, delay-sensitive remote sensing, transmission scheduling, accelerated reinforcement learning, structural properties, value function, post-decision state learning, virtual experience learning.**

Manuscript received May 5, 2019; revised October 3, 2019, December 3, 2019, and January 24, 2020; accepted January 26, 2020. Date of publication February 13, 2020; date of current version February 28, 2020. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Soummya Kar. The work of Nikhilesh Sharma and Nicholas Mastronarde was supported by the National Science Foundation (NSF) under Award ECCS-1711335. The work of Jacob Chakareski was supported in part by the NSF under Awards CCF-1528030, ECCS-1711592, CNS-1836909, and CNS-1821875, and in part by the research gifts and an Adobe Data Science Award from Adobe Systems. *(Corresponding author: Nikhilesh Sharma.)*

Nikhilesh Sharma and Nicholas Mastronarde are with the Department of Electrical Engineering, University at Buffalo, Buffalo, NY 14260 USA (e-mail: nsharma9@buffalo.edu; nmastron@buffalo.edu).

Jacob Chakareski is with the Ying Wu College of Computing, New Jersey Insitute of Technology, Newark, NJ 07103 USA (e-mail: jakov@jakov.org).

Digital Object Identifier 10.1109/TSP.2020.2973125

I. Introduction

ENERGY-CONSTRAINED wireless sensors are increasingly used for emerging latency-sensitive applications such as real-time remote visual sensing [1], [2], body sensor networks [3], smart grid monitoring, UAV-IoT, and mobile virtual and augmented reality [4]–[6]. However, these sensors are subject to time-varying channel conditions and generate stochastic traffic loads – arising due to the compression algorithms that nodes apply before transmitting the sensed data [7] and due to the event-driven nature of many sensor network applications [3], [8] – which makes it very challenging for them to support such applications. This is further complicated by the introduction of wireless sensors powered by energy harvested from the environment (e.g., ambient light, vibration/motion, or RF energy [9]). Although energy harvesting sensors (EHSs) can operate autonomously without access to power lines and without the need to change their batteries, the stochastic nature of the harvested energy sources poses further challenges in sensor power management, transmission power allocation, and transmission scheduling due to the uncertainty in the amount of energy available for communication. Therein arises a need to study and optimize the transmission scheduling policies employed by these sensors.

A. Related Work

A lot of recent work focuses on offline computation of optimal transmission policies for EHSs [10]–[13]. For example, Gurakan and Ulukus [10] consider a multiaccess channel with two EHSs. Assuming that both energy and traffic arrive intermittently over time, and that their arrival processes are known a priori, they derive the optimal offline transmission power and rate allocations that maximize a sum rate objective function. Lu *et al.* [11] formulate a throughput-optimal channel selection policy for EHSs operating as secondary users in a cognitive radio network. Gunduz *et al.* [13] identify Markov decision processes (MDPs [14]) as a useful tool for optimizing EHSs in unpredictable environments with only causal information about the past and present, and statistical information about the future dynamics. Sharma *et al.* [12] formulate both throughput-optimal and delay-optimal energy management policies as MDPs. While these studies identify techniques for calculating optimal transmission policies offline, they do not provide analytical insights on the characteristics of the optimal solutions or methods to solve the considered problems without a priori knowledge of the energy and traffic arrival processes.

Complementing the aforementioned research, another important body of work focuses on characterizing optimal transmission policies for EHSs [3], [15]–[21]. For example, numerous studies have shown that optimal power allocation policies for EHSs are achieved using various water-filling strategies [15]–[17]. Ozel *et al.* [15] consider two related problems: (i) maximizing the number of bits transmitted by a deadline and (ii) minimizing the time to transmit a certain number of bits. They identify that the transmission power over time that optimizes the first objective is achieved through directional water-filling. Ho and Zhang [16] consider the problem of throughput-optimal power allocation over a finite horizon. If unlimited energy can be stored in the battery and full state information is available about past, present, and future slots, they prove that the optimal energy allocation solution is based on water-filling, where the water levels follow a staircase function. Yang and Ulukus [17] consider a two-user multiple access channel. Their goal is to minimize the required time by which all packets from both users are transmitted, by controlling the users' transmission powers and rates. Under the assumption that the energy harvesting times and amounts are known a priori, they prove that the optimal power allocation policy can be found by backward water-filling.

Other characteristics of optimal transmission policies for EHSs have also been studied [7], [18]–[20]. For example, Yang and Ulukus [18] aim to adapt the transmission rate according to traffic load and available energy to minimize the packet delivery time. Assuming prior knowledge of data and energy arrivals, they show that the optimal transmission rates increase in time. Michelusi *et al.* [19] formulate the problem of maximizing the average importance of transmitted data as an MDP. They show that the EHS should only transmit data having importance above a threshold that is a strictly decreasing function of the energy level. Aprem *et al.* [20] formulate outage optimal power control policies for EHSs. For the special case of binary power levels, they show that the optimal policy for the underlying MDP represents a battery state threshold. Zordan *et al.* [7] formulate optimal lossy compression policies for EHSs using constrained MDPs. They demonstrate that the optimal compression policy is non-decreasing in the battery, channel, and energy source states.

In practical scenarios, however, the stochastic processes governing the channel, captured data, and harvested energy dynamics are *unknown a priori*. This necessitates *online learning of transmission scheduling policies* to adapt to the experienced dynamics on-the-fly. In this context, reinforcement learning (RL [22], [23]) is an extremely useful tool. For instance, Blasco *et al.* [24] propose the use of Q-learning [25] (the most widely used RL technique) to maximize the throughput of an energy harvesting transmitter that cannot store data in a buffer. Ortiz *et al.* [26] use an approximate SARSA algorithm with linear function approximation in a point-to-point energy harvesting system with a finite battery to find a power allocation policy that maximizes throughput. While Q-learning and SARSA can solve problems with small action/state spaces, they exhibit very poor convergence rates. This makes them inappropriate for problems with large state spaces or tight timing constraints, such as the one we consider herein. In [27], the authors propose a Bayesian RL approach in an energy harvesting system to decide the transmit power and the number of transmitted data packets that will maximize the long-term expected reward. However, this approach requires keeping track of the number of times that the system transitions from one state to another under each action. This makes it inappropriate for problems with large state spaces and deployment on systems with limited memory.

While the aforementioned work makes great progress towards demonstrating the utility of RL in energy harvesting systems, it is limited to data-driven RL algorithms that do not incorporate useful information from the underlying system model. Exploiting such knowledge about the nature of the available actions (e.g., scheduling actions), the system's dynamics (e.g., packet losses, packet queuing behavior, and battery state evolution), and the system's cost structure (e.g., delay and packet overflows) can significantly increase the learning rate, decrease the complexity, and reduce the memory requirements of RL algorithms, thereby making them suitable for EHSs. We pursue this approach herein.

B. Contributions

We consider the delay-sensitive energy harvesting scheduling (DSEHS) problem where an EHS aims to find the optimal scheduling policy that minimizes the expected queuing delay experienced by its captured data packets. We formulate the DSEHS problem as a discrete-time MDP that takes into account the stochastic captured data traffic loads, harvested energy, and channel dynamics. Our primary contribution is the development of a novel RL framework to solve the DSEHS problem online without a priori knowledge of these dynamics.

We leverage three techniques to accelerate the RL algorithm, while limiting its computational and memory complexity. First, we use *post-decision states* (PDS [22], [23], [28], [29]), which capture the system state after an action is taken, but before the unknown dynamics take place.[1] PDSs allow us to decompose the problem into *known* and *unknown* components, so that the algorithm only needs to learn the latter. This is in contrast to purely data-driven RL algorithms, such as Q-learning, SARSA, and Bayesian RL, which do not take advantage of information that is available about the system model. Using PDSs also exposes that in the DSEHS problem, the system's unknown dynamics are independent of the action variable. This allows us to avoid action exploration [22], which can degrade the run-time performance of RL algorithms [23], [28].[2] Second, we leverage so-called virtual experience (VE [23], [28]), which allows us to update the value function at multiple states in each time slot. VE dramatically improves the learning algorithm's convergence rate at the cost of increased computational complexity. Lastly, we propose a novel value function approximation to reduce the complexity induced by VE and reduce the memory required to store the value function.

Our proposed approximation is motivated by the *structure* of the optimal value function.[3] Specifically, we show that the optimal value function is non-decreasing and has increasing differences in the buffer state and that it is non-increasing and has increasing differences in the battery state. Owing to these

[1]In the DSEHS problem, the unknown dynamics include the number of data packet arrivals, the amount of harvested energy, and the next channel state realization.

[2]In RL, there is an important tradeoff between "exploiting" the action that has the best known expected value and "exploring" alternative actions to discover those with even better values. In practice, action exploration results in frequently taking sub-optimal actions, thereby degrading the system's performance during the learning process.

[3]By *structure* we refer to specific mathematical properties of the optimal value function. In prior literature, structural properties of the optimal value function include, for example, integer convexity/concavity [30] and supermodularity/submodularity [14], [29]. Such properties are important because they can be exploited via *value function approximation* methods, wherein the value function is represented, computed, and/or learned in a memory/computationally efficient manner.

TABLE I
LIST OF NOTATION

Notation	Meaning	Notation	Meaning
n	Time slot index	ΔT	Time slot duration
N_b, N_e, N_h	Total buffer, battery, channel states	a	Scheduling action
$\mathcal{S}_b, \mathcal{S}_e, \mathcal{S}_h, \mathcal{S}$	Buffer, battery, channel, joint state set	l, e_H	Data and energy packet arrivals
b, e, h, s	Buffer, battery, channel, joint state	f	Goodput
$\widetilde{b}, \widetilde{e}, \widetilde{h}, \widetilde{s}$	Post decision states (PDSs)	P_{TX}, e_{TX}	Transmit power and energy
$P^l(l), P^{e_H}(e_H)$	Data and energy arrival distributions	$P^f(f\|a)$	Goodput distribution
$P^b(b'\|[b,h],a)$	Buffer transition probabilities	$P^e(e'\|[e,h],a)$	Battery transition probabilities
$P^h(h'\|h)$	Channel transition probabilities	$P(s'\|s,a)$	State transition probabilities
$c(s,a), c([b,h],a)$	Buffer cost function	q	Packet loss rate
$V(s), \widetilde{V}(s)$	Value function, PDS value function	$Q(s,a)$	Action-value function
$\hat{V}(s)$	Approximated PDS value function	$\pi(s)$	Policy
γ, η	Discount factor, packet drop penalty	α	Learning rate
$\mathcal{T}(h)$	Quadtree in channel state h	δ	Approximation error threshold

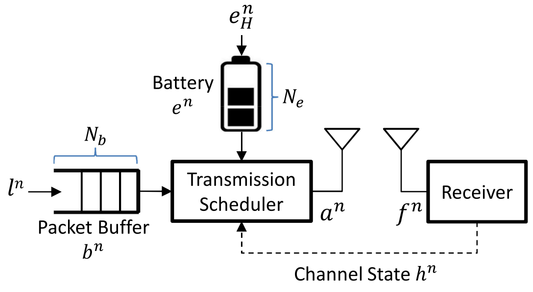


Fig. 1. System block diagram.

structural properties, we are able to develop a value function approximation and corresponding RL algorithm that can quickly learn a near-optimal value function with provably *bounded* and *controllable* error. We refer to this structure-aware algorithm as *grid learning*.

Finally, we demonstrate through MATLAB simulations that our approach achieves competitive performance to a state-of-the-art RL algorithm, at potentially orders of magnitude lower computational and memory complexity. Additionally, it achieves considerable performance gains over Q-learning.

The rest of the paper is organized as follows. We introduce our system model in Section II, formulate the DSEHS problem in Section III, and introduce our RL framework in Section IV. We analyze the structural properties of the DSEHS problem in Section V-A and formulate the proposed structure-aware accelerated RL algorithm in Section V-B. We present our simulation results in Section VI and conclude in Section VII.

II. Delay-Sensitive Energy-Harvesting Model

We consider a time-slotted single-input single-output (SISO) point-to-point wireless communication system in which an energy harvesting sensor transmits latency-sensitive data over a fading channel. The system model is depicted in Fig. 1. The system comprises two buffers: a data packet buffer that can hold up to N_b data packets and an energy buffer (battery) that can hold up to N_e energy packets, where N_b and N_e are possibly infinite. We assume that time is divided into slots with fixed length ΔT (seconds) and that the system's state in the n-th time slot is denoted by $s^n \triangleq (b^n, e^n, h^n) \in \mathcal{S}$, where $b^n \in \mathcal{S}_b = \{0, 1, \ldots, N_b\}$ is the packet buffer state (i.e., the number of data packets in the buffer), $e^n \in \mathcal{S}_e = \{0, 1, \ldots, N_e\}$ is the battery state (i.e., the number of energy packets in the battery), $h^n \in \mathcal{S}_h$ is the channel fading state, and $\mathcal{S} = \mathcal{S}_b \times \mathcal{S}_e \times \mathcal{S}_h$ is the set of system states.

At the beginning of each time slot, the EHS observes the state of the system s^n and takes a scheduling action $a^n \in \mathcal{A}(s) \subseteq \{0, 1, \ldots, N_a\}$, where a^n denotes the number of transmitted packets, $\mathcal{A}(s)$ denotes the set of feasible scheduling actions in state s, and N_a denotes the maximum number of packets that can be transmitted in one time slot. Informally, the goal of the EHS is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$, which maps states to feasible actions in each time slot in order to minimize the average packet queuing delay given the scarce harvested energy. The notation used throughout the paper is summarized in Table I. Note that we omit the time index n from our notation when it is not an essential component of an equation or when we are not referring to a specific value of a variable in a specific time slot. We now introduce the detailed system model before formalizing the problem in Section III.

Channel model: We assume a block-fading channel that is constant during each time slot and may change from one slot to the next. Similar to prior work [7], [23], [31]–[33], we assume that the channel fading coefficient $h^n \in \mathcal{S}_h$ is known to the transmitter at the start of each time slot, that $\mathcal{S}_h$ denotes a finite set of N_h channel states, and that the evolution of the channel state can be modeled as a finite state Markov chain with transition probability function $P^h(h^{n+1}|h^n)$.

Physical layer model: We assume that the physical layer transmits at a data rate β^n/T_s (bits/s), where β^n is the number of bits per symbol determined by the modulation scheme and T_s (s) is the symbol duration. Therefore, in order to transmit a^n packets of size L (bits) in ΔT (s), a modulation scheme must be selected such that

$$\beta^n = \lceil a^n L T_s / \Delta T \rceil \quad \text{(bits/symbol)}, \tag{1}$$

where $\lceil x \rceil$ denotes the ceiling operator, which rounds x up to the nearest integer.

Similarly to [7], [23], [28], we set a target bit error probability (BEP) for all transmissions, which we denote by BEP_{target}. We further assume that, given the channel state and target BEP, the transmission power is a non-decreasing function of the scheduling action a^n, i.e.,

$$P_{\text{TX}}^n = P_{\text{TX}}(h^n, a^n; BEP_{\text{target}}) \quad \text{(Watts)}. \tag{2}$$

This assumption holds for typical modulation schemes, such as M-ary PSK and M-ary QAM [34, Table 6.1]. Also similarly to [7], [23], [28], we do not consider coding; however, it can be integrated by appropriately modifying (1) and (2).

Energy harvesting model: Similar to [24], we assume that battery energy is stored in the form of energy packets. Let $e_H^n \in \mathcal{E} = \{0, 1, \ldots, M_{e_H}\}$ denote the number of energy packets that are available for harvesting in the nth time slot, where M_{e_H} is the maximum number of energy packets that can be harvested in one time slot. Additionally, let $P^{e_H}(e_H)$ denote the energy packet arrival distribution. Intuitively, M_{e_H} and P^{e_H} depend on the technology used (e.g., solar, piezoelectric, or RF), the energy packet size, the time slot duration, and the ambient environment. Energy packets are assumed to arrive at the end of each time slot such that those that arrive in time slot n cannot be used until time slots $n' > n$. Therefore, the battery state at the start of time slot $n+1$ can be found through the following recursion:

$$e^{n+1} = \min(e^n - e_{\text{TX}}^n + e_H^n, N_e), \tag{3}$$

where $e_{\text{TX}}^n = e_{\text{TX}}(h^n, a^n; BEP_{\text{target}})$ denotes the number of energy packets consumed in time slot n given the channel state h^n, scheduling action a^n, and target BEP. For simplicity, we assume that the transmission energy e_{TX}^n is an integer multiple of energy packets, such that

$$\begin{aligned} e_{\text{TX}}^n &= e_{\text{TX}}(h^n, a^n; BEP_{\text{target}}) \\ &= \lceil P_{\text{TX}}(h^n, a^n; BEP_{\text{target}})\Delta T \rceil \text{ (energy packets).} \end{aligned} \tag{4}$$

Note that we only allow transmission actions a^n such that $e_{\text{TX}}(h^n, a^n; BEP_{\text{target}}) \leq e^n$ because the EHS cannot consume more energy than it has available. For notational simplicity, we will omit the transmission energy's dependence on the target BEP in the remainder of the paper.

Given the energy arrival distribution P^{e_H}, the battery state e, and the action a, the probability of observing battery state e' in the next time slot can be computed as:

$$P^e(e'|[e,h],a) = \mathbb{E}_{e_H} \mathbb{I}_{\{e' = \min(e - e_{\text{TX}}(h,a) + e_H, N_e)\}}, \tag{5}$$

where $\mathbb{E}_{e_H}$ denotes the expectation over the harvested energy, and $\mathbb{I}_{\{\cdot\}}$ is an indicator function that is set to 1 when $\{\cdot\}$ is true and is set to 0 otherwise.

Traffic model: Let $l^n \in \mathcal{L} = \{0, 1, \ldots, M_l\}$ denote the number of data packets generated by the sensor in the nth time slot, where M_l denotes the maximum number of packets that can be generated in one time slot. Additionally, let $P^l(l)$ denote the data packet arrival distribution. Intuitively, M_l and P^l depend on the sensing modality, data compression strategy, data packet size, time slot duration, and sensing environment. The buffer state at the start of time slot $n+1$ can be found through the following recursion:

$$b^{n+1} = \min(b^n - f^n + l^n, N_b), \tag{6}$$

where $f^n = f(a^n; BEP_{\text{target}})$ is the number of packets that are correctly received at the receiver in time slot n. By definition, $0 \leq f(a^n; BEP_{\text{target}}) \leq a^n$ because the number of correctly received packets cannot be more than the number of transmitted packets. We also only allow transmission actions a^n such that $a^n \leq b^n$ because the EHS cannot transmit packets that are not in its buffer. Note that new packet arrivals cannot be transmitted in time slot n. Additionally, packets that are not successfully received by the receiver in time slot n remain in the front of the buffer so that they can be retransmitted in a future time slot $n' > n$. Retransmissions are scheduled based on the policy π, just like any other transmission.

Assuming independent and identically distributed (i.i.d.) bit errors, f^n can be modeled as a binomial random variable with conditional probability mass function $P^f(f|a; BEP_{\text{target}}) = \text{Bin}(a, 1-q) = \binom{a}{f}(1-q)^f q^{a-f}$, $f = 0, 1, \ldots, a$, where $q = 1 - (1 - BEP_{\text{target}})^L$ is the packet loss rate (PLR) for a packet of size L (bits). We refer to $P^f(f|a; BEP_{\text{target}})$ as the *goodput distribution* because it denotes the distribution over the number of *correctly received* packets in a time slot. Note that, although we assume that the goodput has a binomial distribution, within our proposed framework it may follow any distribution that depends on the scheduling action and the target BEP. For notational simplicity, hereafter we omit the goodput distribution's dependence on the target BEP.

Given P^f, the arrival distribution P^l, the buffer state b, the action a, and the BEP_{target} (omitted from the notation), the probability of observing buffer state b' in the next time slot can be calculated as follows:

$$P^b(b'|[b,h],a) = \mathbb{E}_{f,l} \mathbb{I}_{\{b' = \min(b - f + l, N_b)\}}. \tag{7}$$

III. The Delay-Sensitive Energy-Harvesting Scheduling (DSEHS) Problem

Let $\pi : \mathcal{S} \to \mathcal{A}$ denote a policy that maps states to actions. The objective of the DSEHS problem is to determine the optimal policy π^* that minimizes the average packet queuing delay given the available energy. However, this does not mean that the policy should greedily transmit packets whenever there is energy to do so. On the contrary, it may be beneficial to abstain from transmitting packets in bad channel states and wait to transmit them in good channel states to avoid wasting scarce harvested energy. At the same time, the policy should not be too conservative. For instance, if the battery is (nearly) full, transmitting packets will make room for more harvested energy, which otherwise would be lost due to the finite battery size. We formulate this challenging sequential-decision problem, in which present scheduling actions affect the EHS's future performance, as an MDP [14].

A. MDP Preliminaries

An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, c, P, \gamma)$, where

- $\mathcal{S}$ is a finite set of states;
- $\mathcal{A}$ is a finite set of actions;
- $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a cost function, which defines the expected cost of taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$;
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition probability function, which defines the probability of transitioning to state $s' \in \mathcal{S}$ after taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$; and
- $\gamma \in [0, 1)$ is a discount factor, which determines how much weight is given to the future costs when deciding which action to take in each state.

A *value function*, $V^\pi(s)$, estimates how good (or bad) it is to be in each state while following the policy π. Formally, $V^\pi(s)$ is defined as,

$$V^\pi(s) = \mathbb{E}\left[\sum_{n=0}^{\infty} (\gamma)^n c(s^n, \pi(s^n)) | s = s^0\right], \tag{8}$$

where $(\gamma)^n$ denotes the discount factor to the nth power and the expectation is taken over the sequence of states, which is governed by the controlled Markov chain with transition probabilities $P(s'|s,a)$. In words, $V^\pi(s)$ is the expected discounted cost when starting in state s and following policy π, where the cost incurred n time slots in the future is discounted by $(\gamma)^n$. We can rewrite $V^\pi(s)$ recursively by taking advantage of the transition probability function:

$$V^\pi(s) = c(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s'). \tag{9}$$

In (9), it is clear that the action $\pi(s)$ taken in state s not only affects the immediate cost, i.e., $c(s, \pi(s))$, but also the expected future cost, i.e., $\sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s')$.

The objective of an MDP is to find a policy that optimizes the value function. Since we aim to minimize the discounted cost, we have:

$$V^*(s) = \min_\pi V^\pi(s), \forall s \in \mathcal{S}. \tag{10}$$

The solution to (10) satisfies the following Bellman equation, $\forall s \in \mathcal{S}$ [14], [22]:

$$V^*(s) = \min_{a \in \mathcal{A}} \left\{ c(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^*(s') \right\} \tag{11}$$

$$= \min_{a \in \mathcal{A}} Q^*(s, a), \tag{12}$$

where V^* is the *optimal value function* and Q^* is the *optimal action-value function*. In words, Q^* is the value obtained by taking action a in state s and then following the optimal policy thereafter. Given V^* or Q^*, the *optimal policy* π^* can be determined by taking the action in each state that minimizes the right-hand side of (11) or (12). We are now ready to formulate the DSEHS problem as an MDP.

B. Formulation of the DSEHS Problem as an MDP

In the DSEHS problem, the set of states $\mathcal{S} = \mathcal{S}_b \times \mathcal{S}_e \times \mathcal{S}_h$ is defined as the Cartesian product of the sets of buffer states $\mathcal{S}_b$, battery states $\mathcal{S}_e$, and channel states $\mathcal{S}_h$. The set of *feasible actions* $\mathcal{A}(b, e, h)$ depends on the buffer, battery, and channel states. Specifically,

$$\mathcal{A}(b, e, h) = \left\{ a \in \{0, \ldots, N_a\} : a \leq b \text{ and } e_{TX}(h, a) \leq e \right\}. \tag{13}$$

This means that the EHS cannot schedule more packets than are available in the buffer (i.e., $a \leq b$) and that there is sufficient energy in the battery to transmit the desired number of packets (i.e., $e_{TX}(h, a) \leq e$).

We define the cost function, denoted here as the *buffer cost*, such that it penalizes large buffer states. Formally, we define the buffer cost as the sum of the *holding cost* and the expected *overflow cost* with respect to the arrival and goodput distributions, i.e.,

$$c([b, h], a) = b + \eta \mathbb{E}_{f,l}\{\max(b - f + l - N_b, 0)\}. \tag{14}$$

Note that the buffer cost does not directly depend on the battery state e, though the set of feasible actions does. The holding cost is simply the buffer state (i.e., the first term on the right-hand side of (14)). Meanwhile, the overflow cost (i.e., the second term on the right-hand side of (14)) imposes a large penalty η for each expected packet overflow, where the expectation is taken over the arrival and goodput distributions (i.e., P^l and P^f, respectively). We discuss our choice of cost function, and its relation to the queuing delay, in more detail in Appendix A.

We define the transition probability function as the product of the conditionally independent buffer state transition probabilities (7), battery state transition probabilities (5), and channel state transition probabilities: i.e.,

$$P(s'|s, a) = P^b(b'|[b, h], a) P^e(e'|[e, h], a) P^h(h'|h). \tag{15}$$

Finally, the DSEHS problem's objective is to find a scheduling policy that minimizes the value function as in (10). The resulting optimal value function satisfies the standard Bellman equation (11), but with the generic cost function $c(s, a)$ and transition probability function $P(s'|s, a)$ replaced respectively with the buffer cost $c([b, h], a)$ defined in (14) and the transition probabilities defined in (15).

As the channel, energy arrival, and traffic arrival dynamics are unknown a priori, the optimal policy cannot be computed offline using standard dynamic programming techniques, such as value iteration and policy iteration [22], [35]. Instead, it must be learned online as the EHS operates in its environment. Existing online approaches in EHSs typically rely on Q-learning [24]. However, Q-learning exhibits extremely slow convergence rates for problems with many states and actions. In our prior work [23], we proposed a fast RL algorithm that achieves up to three orders of magnitude faster convergence rates than Q-learning. However, its complexity is too high for EHSs. In Section V, we adapt the solution in [23] to create a fast and low-complexity RL algorithm based on value function approximation, which is better suited for EHSs. However, before we present the new algorithm, we must review the RL framework developed in [23].

IV. Reinforcement Learning Framework

In this section, we introduce fundamental RL concepts that we build on in Section V-B. In Section IV-A, we review the post-decision state (PDS) concept. In Section IV-B, we describe PDS learning, which learns a value function defined over the PDSs. In Section IV-C, we prove that the PDS learning algorithm converges. In Section IV-D, we introduce the concept of virtual experience.

A. Post-Decision State Based Dynamic Programming

A PDS, denoted by $\tilde{s} \triangleq (\tilde{b}, \tilde{e}, \tilde{h}) \in \mathcal{S}$, is a state of the system after all controllable/known system dynamics have occurred after taking an action, but before the unknown system dynamics occur [22], [23], [31]. Herein,

$$\tilde{s}^n = (\tilde{b}^n, \tilde{e}^n, \tilde{h}^n) = ([b^n - f^n], [e^n - e_{\text{TX}}(h^n, a^n)], h^n) \tag{16}$$

is the PDS in time slot n. The buffer's PDS $\tilde{b}^n = b^n - f^n$ characterizes the buffer state after packets are transmitted (if any), but before any new packets arrive; the battery's PDS $\tilde{e}^n = e^n - e_{\text{TX}}(h^n, a^n)$ characterizes the battery state after energy packets are consumed (if any), but before any new energy packets arrive; and the channel's PDS $\tilde{h}^n = h^n$ is the same as the channel state at time n. In other words, the PDS incorporates all of the known information about the transition from state s^n to state s^{n+1} after taking action a^n. Note that, although the realization of f^n is not known at the time the action is taken, its distribution P^f is known. As will become clear in (19), this is sufficient to include it in the buffer's PDS. Meanwhile, the unknown dynamics in the transition from state s^n to s^{n+1}, i.e., the channel state transition from h^n to $h^{n+1} \sim P^h(\cdot|h^n)$, the data packet arrivals $l^n \sim P^l(\cdot)$, and the energy packet arrivals $e_H^n \sim P^{e_H}(\cdot)$ are not included in the PDS because P^h, P^l, and P^{e_H} are unknown a priori. The next state can be expressed in terms of the PDS as follows:

$$s^{n+1} = (b^{n+1}, e^{n+1}, h^{n+1})$$
$$= \left(\min(\tilde{b}^n + l^n, N_b), \min(\tilde{e}^n + e_H^n, N_e), h^{n+1}\right). \quad (17)$$

Just as we defined a value function over the conventional states, we can define a PDS value function over the PDSs. Let $\tilde{V}^*$ denote the optimal PDS value function. $\tilde{V}^*$ and V^* are related by the following Bellman equations:

$$\tilde{V}^*(\tilde{s}) = \eta \mathbb{E}_l \max(\tilde{b} + l - N_b, 0) + \gamma \mathbb{E}_{l,e_H,h'} V^* \times ([\min(\tilde{b} + l, N_b), \min(\tilde{e} + e_H, N_e), h']) \quad (18)$$

$$V^*(s) = \min_{a \in \mathcal{A}(b,e,h)} \left\{ b + \mathbb{E}_f \tilde{V}^*(b - f, e - e_{TX}(h,a), h) \right\}. \quad (19)$$

Given $\tilde{V}^*(\tilde{s})$, the optimal policy $\pi^*(s)$ can be found by taking the action in each state that minimizes the right-hand side of (19). Note that, since the goodput distribution P^f is known, we can take an expectation of the PDS value function over the buffer's PDS $b - f$ in (19). In Appendix B, we derive (18) and (19), and show that they are equivalent to the standard Bellman equation in (11).

B. Post-Decision State Learning

PDS learning is a stochastic iterative algorithm for learning the PDS value function $\tilde{V}^*(\tilde{s})$ without prior knowledge of the data arrival distribution P^l, energy arrival distribution P^{e_H}, and channel transition probabilities P^h.

PDS learning is presented in Algorithm 1. At the start of time slot n, PDS learning takes the greedy action a^n that minimizes the right-hand side of (20). After observing the unknown dynamics (comprising the data packet arrivals $l^n \sim P^l(\cdot)$, energy packet arrivals $e_H^n \sim P^{e_H}(\cdot)$, and the next channel state $h^{n+1} \sim P^h(\cdot|h^n)$), the algorithm evaluates the PDS $(\tilde{b}^n, \tilde{e}^n, \tilde{h}^n)$ as defined in (16). The core of the PDS learning algorithm is the PDS value function update defined in Algorithm 2 (`update_PDSV`). When `update_PDSV` is called in Algorithm 1, it takes as input the current PDS value function estimate $\tilde{V}^n$, the current PDS $(\tilde{b}^n, \tilde{e}^n, \tilde{h}^n)$, the current realization of the dynamics (l^n, e_H^n, h^{n+1}), and the learning rate parameter $\alpha^n \in [0, 1]$. It then uses (22) to compute a new PDS value function estimate as a weighted average of (i) the current PDS value function estimate $\tilde{V}^n(\tilde{b}^n, \tilde{e}^n, \tilde{h}^n)$ and (ii) a new sample estimate of the PDS value function, i.e., $\eta \max(\tilde{b}^n + l^n - N_b, 0) + \gamma V^n(b^{n+1}, e^{n+1}, h^{n+1})$, derived based on the observed dynamics and the next state's estimated value $V^n(b^{n+1}, e^{n+1}, h^{n+1})$ as computed in (21).

C. The Convergence of Post-Decision State Learning

In this section, we prove that the sequence of PDS value functions $\tilde{V}^n$ generated by the PDS learning algorithm converges to $\tilde{V}^*$ with probability 1 as $n \to \infty$. We begin by introducing the concept of a "well-behaved" stochastic iterative algorithm, which is known to converge under mild conditions [36]. In the remainder of this section, we let $\|X\|$ denote the L_∞ norm of the vector X, i.e., $\|(X(1), X(2), \ldots, X(k))\| = \max_i X(i)$.

Algorithm 1: Post-Decision State Learning.

```
1: initialize $\tilde{V}^0(\tilde{b}, \tilde{e}, \tilde{h}) = 0$ for all $(\tilde{b}, \tilde{e}, \tilde{h}) \in \mathcal{S}$
2: for time slot $n = 0, 1, 2, \ldots$ do
3:     Take the greedy action:
           $a^n = \arg\min_{a \in \mathcal{A}(b^n, e^n, h^n)} \left\{ b^n + \sum_{f=0}^{a} P^f(f|a) \tilde{V}^n(b^n - f, e^n - e_{TX}(h^n, a), h^n) \right\}$   (20)
4:     Observe $l^n$, $e_H^n$, and $h^{n+1}$
5:     Evaluate $\tilde{b}^n$, $\tilde{e}^n$, $\tilde{h}^n$ using (16)
6:     Calculate $\tilde{V}^{n+1}(\tilde{b}^n, \tilde{e}^n, \tilde{h}^n)$ using Algorithm 2:
       update_PDSV $(\tilde{V}^n, [\tilde{b}^n, \tilde{e}^n, \tilde{h}^n], [l^n, e_H^n, h^{n+1}], \alpha^n)$
7: end for
```

Algorithm 2: Post-Decision State Value Function Update (`update_PDSV`).

```
1: input $\tilde{V}$, $[\tilde{b}, \tilde{e}, \tilde{h}]$, $[l, e_H, h']$, and $\alpha$
2: Evaluate $b' = \min(\tilde{b} + l, N_b)$ and
   $e' = \min(\tilde{e} + e_H, N_e)$
3: Evaluate the next state's value:
       $V(b', e', h') = \min_{a \in \mathcal{A}(b', e', h')} \left\{ b' + \sum_{f=0}^{a} P^f(f|a) \times \tilde{V}(b' - f, e' - e_{TX}(h', a), h') \right\}$   (21)
4: Update the PDS value function:
       $\tilde{V}(\tilde{b}, \tilde{e}, \tilde{h}) \leftarrow (1 - \alpha)\tilde{V}(\tilde{b}, \tilde{e}, \tilde{h}) + \alpha[\eta \max(\tilde{b} + l - N_b, 0) + \gamma V(b', e', h')]$   (22)
5: return $\tilde{V}(\tilde{b}, \tilde{e}, \tilde{h})$
```

Consider a stochastic iterative algorithm of the form:

$$X^{n+1}(i) = (1 - \beta^n)X^n(i) + \beta^n[(H^n X^n)(i) + w^n(i)], \quad (23)$$

where w^n is a bounded random variable with zero expectation and H^n belongs to a family of contraction mappings. The iteration in (23) constitutes a well-behaved stochastic algorithm if it satisfies the following conditions:

Definition 1: (Well-behaved stochastic iterative algorithm [36]): A stochastic iterative algorithm is well-behaved if:

1) *Stochastic approximation:* The non-negative step sizes β^n satisfy $\sum_{n=0}^{\infty} \beta^n = \infty$ and $\sum_{n=0}^{\infty} (\beta^n)^2 \leq \infty$.
2) *Bounded noise:* There exists a constant G that bounds $w^n(i)$ for any history F^n, i.e., $|w^n(i)| \leq G, \forall n, i$.
3) *Contraction mapping:* There exists a $\gamma \in [0, 1)$ and a vector X^* such that for any X we have $||H^n X - X^*|| \leq \gamma ||X - X^*||$.

Proposition 1: The PDS learning algorithm defined in Algorithm 1 is a well-behaved stochastic iterative algorithm.

Proof: The proof is given in Appendix C. ■

Note that PDS learning converges relatively slowly because it only updates the value of one PDS in each time slot. In the next subsection, we introduce the concept of *virtual experience*,

Algorithm 3: Virtual Experience Learning ($T = 1$).

```
1: initialize $\tilde{V}^0(\tilde{b}, \tilde{e}, \tilde{h}) = 0$ for all $(\tilde{b}, \tilde{e}, \tilde{h}) \in \mathcal{S}$
2: for time slot $n = 0, 1, 2, \ldots$ do
3:     Take the greedy action:
       $$a^n = \underset{a \in \mathcal{A}(b^n, e^n, h^n)}{\arg\min} \left\{ b^n + \sum_{f=0}^{a} P^f(f|a) \tilde{V}^n(b^n - f, e^n - e_{TX}(h^n, a), h^n) \right\} \quad (24)$$
4:     Observe $l^n$, $e_H^n$, and $h^{n+1}$
5:     for all $(\tilde{b}, \tilde{e}) \in \mathcal{S}_b \times \mathcal{S}_e$ do
6:     Calculate $\tilde{V}^{n+1}(\tilde{b}, \tilde{e}, h^n)$ using Algorithm 2:
       update_PDSV $(\tilde{V}^n, [\tilde{b}, \tilde{e}, h^n], [l^n, e_H^n, h^{n+1}], \alpha^n)$
7:     end for
8: end for
```

which allows us to update multiple PDSs in each time slot thereby dramatically improving the convergence rate.

D. Virtual Experience Learning

Virtual experience learning is a state-of-the-art reinforcement learning algorithm that we proposed in our prior work [23]. The key idea is that it is possible to update the value of multiple PDSs in each time slot. In the DSEHS problem, virtual experience learning is enabled by the fact that the unknown data arrival and energy packet arrival dynamics (i.e., $l^n \sim P^l(\cdot)$ and $e_H^n \sim P^{e_H}(\cdot)$, respectively) are independent of the post-decision buffer and battery states (i.e., $\tilde{b}^n$ and $\tilde{e}^n$, respectively). This enables us to update all PDSs with the same post-decision channel state $\tilde{h}^n$, but with different $\tilde{b}$ and $\tilde{e}$ given the observations of l^n, e_H^n, and h^{n+1}. Updating $|\mathcal{S}_b \times \mathcal{S}_e|$ PDSs in every time slot significantly improves the convergence rate at the cost of increased computational complexity. Specifically, if the update is applied every T time slots, then the average number of PDSs updated in each time slot is $|\mathcal{S}_b \times \mathcal{S}_e|/T$. Algorithm 3 provides pseudo-code for virtual experience learning with an update period of $T = 1$.

V. Value Function Approximation-Based Reinforcement Learning

Virtual experience learning is too complex to implement on EHSs because it requires updating $|\mathcal{S}_b \times \mathcal{S}_e|$ PDSs every T time slots. Although T can be increased to further reduce the average learning complexity, this comes at the expense of a significant decrease in the convergence rate [23]. In this section, we pursue a more effective approach to reduce the complexity of virtual experience learning, while still reaping its benefits. Specifically, we propose to learn an *approximate* value function instead of the true value function. To this end, we first present several structural properties of the optimal PDS value function $\tilde{V}^*(s)$ (Section V-A). Then, motivated by these properties, we propose a novel RL algorithm that learns a near-optimal piece-wise planar approximation of the PDS value function (Section V-B).

A. Structural Properties of the Optimal Value Function

Integer convexity is key to understanding the structure of the optimal PDS value function.

Definition 2. (Integer Convex): An integer convex function $f(n) : \mathcal{N} \to \mathbb{R}$ on a set of integers $\mathcal{N} \in \{0, 1, \ldots, N\}$ is a function that has increasing differences in n, i.e.,

$$f(n_1 + m) - f(n_1) \leq f(n_2 + m) - f(n_2) \quad (25)$$

for $n_1 < n_2$ and $n_1, n_2, n_1 + m, n_2 + m \in \mathcal{N}$.

The following propositions establish the key structural properties of the PDS value function with respect to the post-decision buffer state $\tilde{b}$ and the post-decision battery state $\tilde{e}$, respectively. The proofs are given in [37].

Proposition 2: The optimal PDS value function $\tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h})$ has the following structural properties with respect to $\tilde{b}$:

1) $\tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h})$ is non-decreasing in $\tilde{b}$, i.e.,
$$\tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h}) \leq \tilde{V}^*(\tilde{b} + 1, \tilde{e}, \tilde{h}).$$
2) $\tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h})$ has increasing differences in $\tilde{b}$, i.e.,
$$\tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h}) - \tilde{V}^*(\tilde{b} - 1, \tilde{e}, \tilde{h}) \leq \tilde{V}^*(\tilde{b} + 1, \tilde{e}, \tilde{h}) - \tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h}).$$

Proposition 3: The optimal PDS value function $\tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h})$ has the following structural properties with respect to $\tilde{e}$:

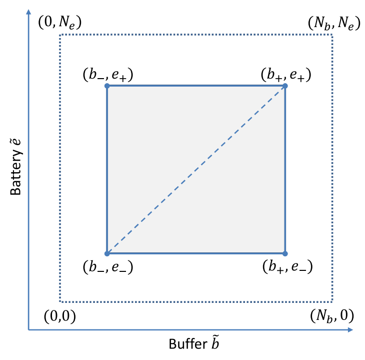
1) $\tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h})$ is non-increasing in $\tilde{e}$, i.e.,
$$\tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h}) \geq \tilde{V}^*(\tilde{b}, \tilde{e} + 1, \tilde{h}).$$
2) $\tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h})$ has increasing differences in $\tilde{e}$, i.e.,
$$\tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h}) - \tilde{V}^*(\tilde{b}, \tilde{e} - 1, \tilde{h}) \leq \tilde{V}^*(\tilde{b}, \tilde{e} + 1, \tilde{h}) - \tilde{V}^*(\tilde{b}, \tilde{e}, \tilde{h}).$$

Proposition 2 implies that the cost to serve an additional data packet increases with the buffer state. Proposition 3 implies that the benefit of an additional energy packet decreases with the available battery energy.

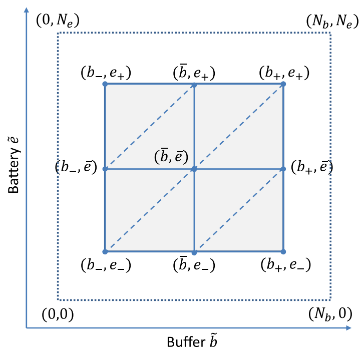
B. Grid Learning

This subsection is organized as follows. We motivate our proposed value function approximation, which is based on a quadtree data structure, in Section V-B1; we formalize the quadtree data structure and present relevant quadtree operations in Section V-B2; we present pseudocode for our proposed reinforcement learning algorithm (called *grid learning*) in Section V-B3; we detail how the value function can be approximated from the quadtree in Section V-B4; and we show how to adaptively refine the quadtree to meet a target error tolerance in Section V-B5.

1) Choice of Function Approximator: Different types of function approximators, including linear function approximators and non-linear function approximators (e.g., neural networks), have been used to approximate value functions. Unfortunately, they lack some desirable properties. Though linear function approximations are guaranteed to converge, there is no way to systematically bound or control the approximation error [22, Section 9.4]. Non-linear function approximations, on the other hand, may converge to *local* minima, but are not guaranteed to do so. In fact, in many cases, they are unstable and fail to converge, even to within a bounded distance from the optimal value function [22, Section 9.2].



(a) Quadtree bounding box.



(b) Quadtree subdivide operation.

Fig. 2. Quadtree construction. Each leaf node is divided into a NW and SE triangle in the piece-wise planar approximation.

Motivated by the fact that the optimal PDS value function has increasing differences in the post-decision buffer and battery states (see Propositions 2 and 3), we propose to approximate it as a piece-wise planar function. We select this specific function approximation because, owing to the PDS value function's aforementioned structural properties, it allows us to develop a low-complexity reinforcement learning algorithm that can quickly learn a near-optimal PDS value function with provably *bounded* and *controllable* error. We refer to this structure-aware algorithm as *grid learning*.

For each post-decision channel state $\tilde{h} \in \mathcal{S}_h$, grid learning constructs a two-dimensional grid of post-decision buffer and battery states on which to learn the PDS value function. However, rather than using a uniform grid, we propose to use a *quadtree* data structure.[4] We select this data structure so that the value function approximation can be adaptively refined in space (i.e., on the buffer-battery plane) and in time (i.e., from slot-to-slot) to meet a predetermined approximation error threshold, δ, while requiring less memory than a uniform grid. To construct the proposed piece-wise planar approximation from the quadtree, we divide each leaf node in the quadtree into two triangles, which lie on two intersecting planes. Together, the planes of all leaf nodes compose the proposed piece-wise planar approximation (see Fig. 2).

2) Quadtree Definition: Let $\mathcal{T}$ denote a quadtree defined on the set of buffer-battery state pairs $\mathcal{S}_b \times \mathcal{S}_e$ within a bounding box (BB) defined as follows (see Fig. 2(a)):

$$\mathtt{BB}(\mathcal{T}) = \{(b_-, e_-), (b_+, e_-), (b_-, e_+), (b_+, e_+)\}, \tag{26}$$

where $0 \leq b_- < b_+ \leq N_b$ and $0 \leq e_- < e_+ \leq N_e$. In words, $\mathtt{BB}(\mathcal{T})$ comprises the extreme vertices of the quadtree. We say that (b, e) lies inside $\mathcal{T}$'s bounding box if $b_- \leq b \leq b_+$ and $e_- \leq e \leq e_+$; otherwise, (b, e) lies outside $\mathcal{T}$'s bounding box.

If $\mathcal{T}$ is a leaf node, then it can be subdivided into four sub-quadtrees (children) spanning its northwest (NW), northeast (NE), southwest (SW), and southeast (SE) quadrants, i.e., $\mathtt{subdivide}\ (\mathcal{T}) = \{\mathcal{T}_{NW}, \mathcal{T}_{NE}, \mathcal{T}_{SW}, \mathcal{T}_{SE}\}$, with bounding boxes defined as follows (see Fig. 2(b)):

$$\mathtt{BB}(\mathcal{T}_{NW}) = \{(b_-, \bar{e}), (\bar{b}, \bar{e}), (b_-, e_+), (\bar{b}, e_+)\},$$

$$\mathtt{BB}(\mathcal{T}_{NE}) = \{(\bar{b}, \bar{e}), (b_+, \bar{e}), (\bar{b}, e_+), (b_+, e_+)\},$$

$$\mathtt{BB}(\mathcal{T}_{SW}) = \{(b_-, e_-), (\bar{b}, e_-), (b_-, \bar{e}), (\bar{b}, \bar{e})\},$$

$$\mathtt{BB}(\mathcal{T}_{SE}) = \{(\bar{b}, e_-), (b_+, e_-), (\bar{b}, \bar{e}), (b_+, \bar{e})\},$$

where $\bar{b} = \lfloor \frac{b_+ + b_-}{2} \rfloor \in \mathcal{S}_b$, $\bar{e} = \lfloor \frac{e_+ + e_-}{2} \rfloor \in \mathcal{S}_e$, and $\lfloor x \rfloor$ is the floor operator, which rounds x down to the nearest integer. With a slight abuse of notation, we write $(b, e) \in \mathcal{T}$ if (b, e) is an element of $\mathcal{T}$'s bounding box or one of its children's bounding boxes, recursively down to its leaf nodes.

3) Grid Learning: Let $\mathcal{T}^n(\tilde{h})$ denote the quadtree used to approximate the PDS value function in channel state $\tilde{h}$ in time slot n. We assume that $\mathtt{BB}(\mathcal{T}^n(\tilde{h}))$ is defined as in (26) for all n. Note that, as in Fig. 2(a), we do not require $\mathcal{T}^n(\tilde{h})$ to span the entire buffer-battery plane (i.e., for $b_- = 0$, $b_+ = N_b$, $e_- = 0$, and $e_+ = N_e$) because N_b and N_e may be very large (or infinite) and it is often unnecessary to closely approximate the value at the extremes of the state space (e.g., if there is an abundant energy supply or very little data to serve).

Grid learning approximates the value of any PDS pair $(\tilde{b}, \tilde{e}) \notin \mathcal{T}^n(\tilde{h})$ using the values of PDS pairs $(\tilde{b}, \tilde{e}) \in \mathcal{T}^n(\tilde{h})$. That is, instead of operating directly on $\tilde{V}$, it operates on an approximate PDS value function $\hat{V}$ such that

$$\hat{V}^n(\tilde{b}, \tilde{e}, \tilde{h}) = \begin{cases} \tilde{V}^n(\tilde{b}, \tilde{e}, \tilde{h}), & \text{if } (\tilde{b}, \tilde{e}) \in \mathcal{T}^n(\tilde{h}) \\ \mathtt{approx}\big(\tilde{V}^n, \mathcal{T}^n(\tilde{h}), [\tilde{b}, \tilde{e}, \tilde{h}]\big), & \text{otherwise.} \end{cases} \tag{27}$$

In Section V-B4, we describe how the function $\mathtt{approx}$ calculates the approximate value of any buffer-battery state pair $(\tilde{b}, \tilde{e}) \notin \mathcal{T}^n(\tilde{h})$

Pseudocode for grid learning with update period $T = 1$ is provided in Algorithm 4. At the start of the algorithm ($n = 0$), we initialize $\mathcal{T}^0(\tilde{h})$ with $\mathtt{BB}(\mathcal{T}^0(\tilde{h}))$ defined as in (26) and initialize its child nodes to empty. In other words, $\mathcal{T}^0(\tilde{h})$ serves as the root of the quadtree and provides the minimum set of grid points from which we can estimate the values of all $(\tilde{b}, \tilde{e}) \in \mathcal{S}_b \times \mathcal{S}_e$ using the proposed piece-wise planar approximation. After initialization, the algorithm proceeds similarly to virtual experience learning (Algorithm 3) with three key differences. First, as noted above, the algorithm operates on an approximate PDS value function $\hat{V}$ instead of the actual PDS value function $\tilde{V}$.[5]

[4]A quadtree is a tree data structure in which each node has exactly four children, or has no children. A node that does not have any children associated with it is known as a *leaf* node. Quadtrees are often used to partition a two-dimensional space by recursively subdividing it into four equal quadrants, sub-quadrants, and so on.

[5]In Algorithm 4 we slightly abuse the notation when we use $\hat{V}^n$ on the right-hand side of (30) and as an argument to the $\mathtt{update\_PDSV}$ function. In practice, we have chosen to calculate values of $\hat{V}^n$ on-demand using the $\mathtt{approx}$ function. In this way, we do not need to maintain a full tabular representation of the (approximate) value function.

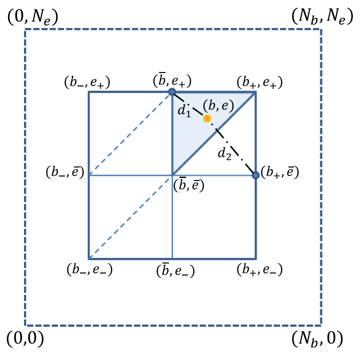


Fig. 3. Associating the buffer-battery state pair (b, e) with the leaf node's closest triangle. If $d_1 < d_2$, then we use the NW triangle; otherwise, we use the SE triangle.

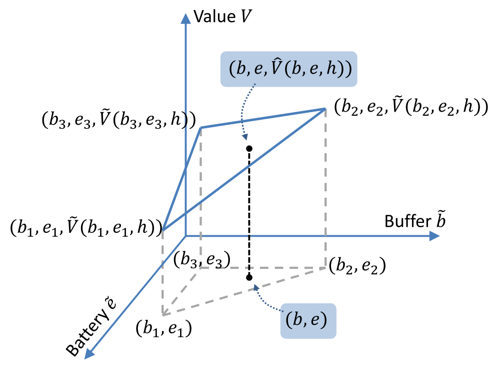


Fig. 4. Calculating the approximate value $\hat{V}(b, e, h)$ of a buffer-battery state pair $(b, e) \notin \mathcal{T}(h)$ using a piece-wise planar approximation. $\hat{V}(b, e, h)$ is calculated as in (28).

Second, the function `update_PDSV` is only called for PDS pairs $(\tilde{b}, \tilde{e}) \in \mathcal{T}^n(\tilde{h})$, rather than all PDS pairs $(\tilde{b}, \tilde{e}) \in \mathcal{S}_b \times \mathcal{S}_e$. Since $\mathcal{T}^n(\tilde{h})$ is only a small subset of $\mathcal{S}_b \times \mathcal{S}_e$ and $\tilde{V}$ is only defined on $\mathcal{T}^n(\tilde{h}), \forall \tilde{h} \in \mathcal{S}_h$, grid learning requires significantly less computation and memory than exhaustive virtual experience learning operating on the full PDS value function (i.e., Algorithm 3). Third, since the approximate value function $\hat{V}$ may not approximate all PDS pairs $(\tilde{b}, \tilde{e}) \in \mathcal{S}_b \times \mathcal{S}_e$ within the target error tolerance δ, we use the `update_grid` function (Algorithm 6) to adaptively refine the approximation over time. We now describe the `approx` and `update_grid` functions in detail.

4) PDS Value Function Approximation: Assume $\mathcal{T}(h)$ is the root of the quadtree and that we want to find the approximate value $\hat{V}(b, e, h)$ of the buffer-battery state pair (b, e), which lies inside of $\mathcal{T}(h)$'s bounding box as defined in (26). The function `approx` achieves this in roughly four steps: 1) associate (b, e) with one of the quadtree's leaf nodes; 2) further associate (b, e) with the leaf node's NW or SE triangle; 3) find the equation of the plane defined by the selected triangle's vertices (hereafter, we will refer to this as the *approximating plane*); and 4) calculate the approximate value of $\hat{V}(b, e, h)$ from the approximating plane.

To be precise, we associate (b, e) with the quadtree's nearest leaf node using a recursive search from the root. We then associate (b, e) with the leaf node's nearest triangle as illustrated in Fig. 3. Specifically, let d_1 and d_2 denote the distances between (b, e) and the leaf node's NW and SE vertices, respectively. If $d_1 < d_2$, then we associate (b, e) with the NW triangle; otherwise, we associate it with the SE triangle.

Denote the vertices of the selected triangle by $\mathbf{x}_i = (b_i, e_i, \tilde{V}(b_i, e_i, h))$, for $i = 1, 2, 3$, as illustrated in Fig 4. These three points define a plane with normal vector $\mathbf{n} = (n_1, n_2, n_3) = (\mathbf{x}_1 - \mathbf{x}_2) \times (\mathbf{x}_1 - \mathbf{x}_3)$, where $\times$ denotes the cross product. The equation of the approximating plane can therefore be written as:

$$n_1(\tilde{b} - b_1) + n_2(\tilde{e} - e_1) + n_3(V - \tilde{V}(b_1, e_1, h)) = 0.$$

Finally, substituting (b, e) for $(\tilde{b}, \tilde{e})$ and solving for V we get:

$$V = \hat{V}(b, e, h) = \tilde{V}(b_1, e_1, h) - \frac{n_1(b - b_1) + n_2(e - e_1)}{n_3}. \tag{28}$$

Pseudocode for the function `approx` is given in Algorithm 5.

The following proposition shows that the maximum error resulting from a piece-wise planar approximation of the optimal PDS value function is bounded.

Proposition 4: Let $\tilde{V}^*$ denote the optimal PDS value function that satisfies the Bellman equation (18). Let $\hat{V}$ denote the approximate PDS value function (27). Let (b, e) be associated with the triangle with vertices $\mathbf{x}_i = (b_i, e_i, \tilde{V}^*(b_i, e_i, h))$, for $i = 1, 2, 3$. It follows that

$$\hat{V}(b, e, h) - \tilde{V}^*(b, e, h) \leq \max_{i\in\{1,2,3\}} \tilde{V}^*(b_i, e_i, h) - \min_{i\in\{1,2,3\}} \tilde{V}^*(b_i, e_i, h). \tag{29}$$

Proof: The result follows from the PDS value function's structural properties that are given in Propositions 2 and 3. In particular, since $\tilde{V}^*$ has increasing differences in $\tilde{b}$ and $\tilde{e}$, the plane defined by the approximating triangle provides an upper bound on the true value function. Additionally, since $\tilde{V}^*$ and $\hat{V}$ are non-decreasing in $\tilde{b}$ and non-increasing in $\tilde{e}$, they are both bounded by $\min_{i\in\{1,2,3\}} \tilde{V}^*(b_i, e_i, h)$ and $\max_{i\in\{1,2,3\}} \tilde{V}^*(b_i, e_i, h)$ for all (b, e) that lie in the approximating triangle. The result in (29) immediately follows. ■

5) Dynamic Grid Update: The function `update_grid` adaptively refines the piecewise-planar approximation until a predetermined maximum error threshold, δ, is met. The algorithm finds the error δ_ℓ among all leaf nodes $\mathcal{T}_\ell \in$ `leaves` $(\mathcal{T})$, where δ_ℓ is calculated as the error defined on the right-hand side of (29). Subsequently, if $\max_\ell \delta_\ell > \delta$, then $\mathcal{T}_\ell$ is subdivided as described in Section V-B2. Pseudocode for the function `update_grid` is given in Algorithm 6.

C. Complexity Analysis

Table II compares the action selection, learning update, and grid update complexities of the proposed *grid learning* algorithm (Algorithm 4), *Approximate SARSA* [26], *Q-learning* [24], *PDS learning* (Algorithm 1) and *virtual experience learning* (Algorithm 3) in each time slot. Note that the grid update complexity is only defined for grid learning because the other algorithms do not include a grid update step. In what follows, let $|\mathcal{S}|$ and $|\mathcal{A}|$ denote the set of states and actions, respectively; let $|\mathcal{S}_b|$, $|\mathcal{S}_e|$ and $|\mathcal{S}_h|$ denote the number of buffer, battery, and channel states, respectively; and let $|\mathcal{L}|$, $|\mathcal{E}|$ and $|\mathcal{F}|$ denote the size of supports for the data packet arrival, energy packet arrival, and goodput distributions, respectively.

TABLE II
ACTION-SELECTION AND ITERATION COMPLEXITY OF SEVERAL LEARNING ALGORITHMS

Algorithm	Action Selection Complexity	Iteration Complexity	Grid Update Complexity
Q-Learning	$\mathcal{O}(\|\mathcal{A}\|)$	$\mathcal{O}(\|\mathcal{A}\|)$	-
Approximate SARSA	$\mathcal{O}(\|\mathcal{A}\|)$	$\mathcal{O}(\|\mathcal{A}\|)$	-
PDS Learning	$\mathcal{O}(\|\mathcal{F}\|\|\mathcal{A}\|)$	$\mathcal{O}(\|\mathcal{F}\|\|\mathcal{A}\|)$	-
Virtual Experience Learning	$\mathcal{O}(\|\mathcal{F}\|\|\mathcal{A}\|)$	$\mathcal{O}(\|\Pi\|\|\mathcal{F}\|\|\mathcal{A}\|)$	-
Grid Learning	$\mathcal{O}(k\|\mathcal{F}\|\|\mathcal{A}\|)$	$\mathcal{O}(k\|\mathcal{T}\|\|\mathcal{F}\|\|\mathcal{A}\|)$	$k\|\mathcal{T}\|$

Algorithm 4: Grid Learning (Update Period $T = 1$).

```
1: initialize $\mathcal{T}^0(\tilde{h})$ for all $\tilde{h} \in \mathcal{S}_h$ as in (26),
   $\tilde{V}^0(\tilde{b}, \tilde{e}, \tilde{h}) = 0$ for all $(\tilde{b}, \tilde{e}) \in \mathcal{T}^0(\tilde{h})$ and $\tilde{h} \in \mathcal{S}_h$, and $\delta$
   to the desired error threshold
2: for time slot $n = 0, 1, 2, \ldots$ do
3:    Take the greedy action:
      $$a^n = \underset{a \in \mathcal{A}(b^n, e^n, h^n)}{\arg\min} \Big\{ b^n + \sum_{f=0}^{a} P^f(f|a) \hat{V}^n(b^n - f, e^n - e_{TX}(h^n, a), h^n) \Big\} \quad (30)$$
4:    Observe $l^n$, $e_H^n$, and $h^{n+1}$
5:    for all $(\tilde{b}, \tilde{e}) \in \mathcal{T}^n(h^n)$ do
6:       Calculate $\tilde{V}^{n+1}(\tilde{b}, \tilde{e}, h^n)$ using Algorithm 2:
         update_PDSV$(\hat{V}^n, [\tilde{b}, \tilde{e}, h^n], [l^n, e_H^n, h^{n+1}], \alpha^n)$
7:    end for
8:    Determine $\mathcal{T}^{n+1}(h^n)$ using Algorithm 6:
      $\mathcal{T}^{n+1}(h^n) \leftarrow$ update_grid$(\tilde{V}^n, \mathcal{T}^n(h^n), \delta)$
9: end for
```

Algorithm 5: Approximate the PDS Value Function (Approx).

```
1: input $\tilde{V}$, $\mathcal{T}$, and $(b, e)$
2: Associate $(b, e)$ with $\mathcal{T}$'s nearest leaf node using a
   recursive search
3: Further associate $(b, e)$ with the leaf node's closest
   triangle as in Fig. 3
4: Calculate $\hat{V}(b, e, h)$ from $\tilde{V}$ as in (28)
5: return $\hat{V}(b, e, h)$
```

Q-learning and Approximate SARSA have action selection and learning update complexities of $\mathcal{O}(|\mathcal{A}|)$ as both algorithms require evaluating the action-value function for all possible actions during their action selection and learning update steps.

The PDS learning's action selection complexity is $\mathcal{O}(|\mathcal{F}||\mathcal{A}|)$ as, from (20), it needs to iterate over all possible goodputs, to compute the right-hand side for one given action, and then over all possible actions, to determine the best action. Its learning update complexity as calculated from (21) is also $\mathcal{O}(|\mathcal{F}||\mathcal{A}|)$ for similar reasons.

Virtual experience learning's action-selection complexity is the same as that of PDS learning, i.e., $\mathcal{O}(|\mathcal{F}||\mathcal{A}|)$. To compute its learning update complexity, we define $|\Pi| = |\mathcal{S}_b \times \mathcal{S}_e|$, which denotes the total number of buffer-battery state pairs. Since virtual experience learning proceeds similar to PDS learning,

Algorithm 6: Dynamic Grid Update (update_grid).

```
1: input $\tilde{V}$, $\mathcal{T}$, and $\delta$
2: for each leaf $\mathcal{T}_\ell \in$ leaves$(\mathcal{T})$ do
3:    Calculate the approximation error $\delta_\ell$ as in (29)
4: end for
5: $\delta_{\max} \leftarrow \max_\ell \delta_\ell$ and $\ell_{\max} \leftarrow \arg\max_\ell \delta_\ell$
6: if $\delta_{\max} > \delta$ then
7:    Subdivide quadtree $\mathcal{T}_{\ell_{\max}}$ and add to $\mathcal{T}$
8: end if
9: return $\mathcal{T}$
```

but updates all buffer-battery pairs in each iteration, its per-step learning update complexity is $\mathcal{O}(|\Pi||\mathcal{F}||\mathcal{A}|)$.

The proposed *grid learning* algorithm features similar complexity to virtual experience learning, save for the differences mentioned in Section V-B4. Thus, the space of points directly evaluated is reduced to the quadtree, $\mathcal{T}$. Additionally, Algorithm 5 introduces a worst-case complexity of $\mathcal{O}(k)$ to determine the approximate value of a $(\tilde{b}, \tilde{e})$ pair in a quadtree with maximum depth k. Thus, its per-iteration complexity is $\mathcal{O}(k|\mathcal{T}||\mathcal{F}||A|)$, and the additional complexity per call of the update_grid method is $\mathcal{O}(k|\mathcal{T}|)$ to check if the quadtree needs to be subdivided further to meet the target error, δ.

VI. SIMULATION RESULTS

We now present our simulation results. In Section VI-A, we describe the simulation setup. In Section VI-B, we compare the proposed grid learning algorithm against Approximate SARSA [26], Q-learning [24], PDS learning, virtual experience learning, and the optimal policy. Finally, in Section VI-C, we explore how the approximation error threshold affects learning performance and study the behavior of our adaptive grid refinement algorithm.

A. Simulation Setup

The simulation parameters used in our MATLAB-based simulator are described in Table III. For illustration, we assume that the buffer and battery have sizes $N_b = 32$ data packets and $N_e = 32$ energy packets, respectively, and that there are $N_h = 8$ channel states with SNRs $h = \{-13, -8.47, -5.41, -3.28, -1.59, -0.08, 1.42, 3.18\}$ dB, similar to [31]. This yields a large state space comprising a total of $(N_b + 1) \times (N_e + 1) \times N_h = 8712$ states. We assume that the channel fading state is known to the transmitter at the beginning of each time slot; however, the Markovian channel transition probability function, P^h, is unknown a priori. We further assume that the data and energy packet arrival distributions, P^l and P^{e_H}, respectively, are Bernoulli, but are unknown a priori. Finally, we set the

TABLE III
SIMULATION PARAMETERS

Parameter	Value	Parameter	Value
Packet Buffer Size, N_b	32	Transmit Action, a	$\{0, 1, 2, 3\}$
Battery Size, N_e	32	Energy Packet Size	2.72 nJ
Modulations	BPSK, 4-PSK, 8-PSK	Discount Factor, γ	0.98
Noise Spectral Density N_0	-174 dBm	Simulation Duration (slots)	50,000
Packet Size, L	127 bytes	Time Slot Duration, ΔT	5 ms
VE Update Period, T_{VE}	10	Grid Update Period, T_{grid}	$\{10, 50, 100\}$
Data Arrival Distribution $P^l(l)$	$\text{Bern}(p), p \in \{0.1, 0.2, \ldots, 0.6\}$	Approximation Error Threshold, δ	$\{5, 7.5, 10, \ldots, 45\}$
Energy Arrival Distribution $P^{e_H}(e_H)$	Bern(0.7)	Packet Overflow Penalty, η	50
Channel States, $h \in \mathcal{S}_h$ (dB) [31]	{-13,-8.47,-5.41,-3.28, -1.59,-0.08, 1.42,3.18}	BEP_{target}	9.89×10^{-5}

discount factor $\gamma = 0.98$ to balance present and expected future costs and to optimize the long term behavior of the scheduling policy. For our energy model, we consider M-PSK modulation. In channel state h, the energy required to transmit a packets with target BEP, BEP_{target}, is given by (4), and the transmission power P_{TX} is approximated by [34, Table 6.1]:

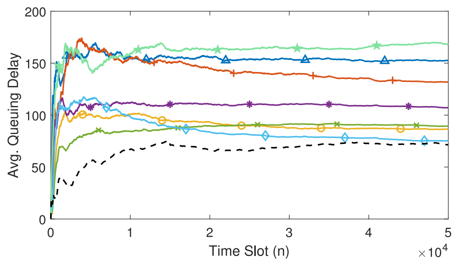
$$P_{\text{TX}}(h, a; BEP_{\text{target}}) = \begin{cases} \frac{N_0}{2\,hT_s}[Q^{-1}(BEP_{\text{target}})]^2, & \text{if } \beta = 1, \\ \frac{N_0}{2\,h\beta T_s \sin^2(\pi/M)}[Q^{-1}(\frac{\beta}{2} \cdot BEP_{\text{target}})]^2, & \text{if } \beta > 1, \end{cases} \tag{31}$$

where $Q^{-1}(\cdot)$ is the inverse of the Q-function; N_0 is the noise spectral density; and β is the number of bits per symbol, which is determined using (1). The energy packet size is selected to be the amount of energy required to transmit one packet in the best channel state. Finally, the BEP target is set to achieve a 1% PLR for packets with size $L = 127$ bytes.

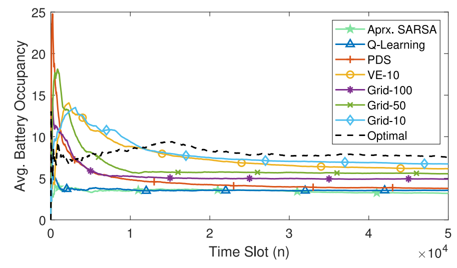
B. Learning Algorithm Comparison

We implement Approximate SARSA as described in [26], Q-learning as described in [22], PDS learning as described in Section IV-B and Algorithm 1, VE learning as described in Section IV-D and Algorithm 3, and grid learning as described in Section V-B and Algorithm 4. Simulation results using the parameters in Table III are presented in Fig. 5, where each curve is obtained by averaging over eight 50,000 time slot simulations with data packet arrival distribution $P^l(l) = \text{Bern}(0.4)$, energy packet arrival distribution $P^{e_H}(e_H) = \text{Bern}(0.7)$, and initial states $b^0 = e^0 = 0$.[6] For grid learning, we use $\delta = 10$ as the approximation error threshold.

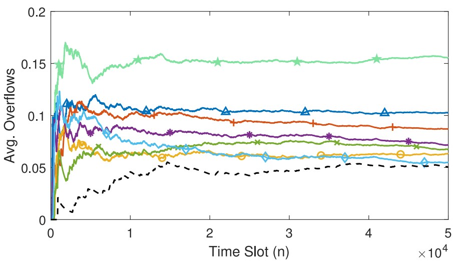
In Fig. 5, the curves labeled "Grid-T" are obtained using grid learning with updates every $T = 10, 50, 100$ time slots; the curve labeled "VE-10" is obtained using VE learning with updates every $T = 10$ time slots; the curves labeled "Q-learning" and "PDS" are obtained from Q-learning and PDS learning, respectively, with updates every time slot; and the curves labeled "Optimal" are obtained from the optimal policy, which is computed offline using *value iteration* [22] assuming that P^h, P^l, and P^{e_H} are known a priori. Fig. 5(a) illustrates the cumulative average queuing delay[7] versus time; Fig. 5(b) illustrates the cumulative average battery occupancy versus time; and Fig. 5(c) illustrates the cumulative average buffer overflows versus time.



(a) Average Queuing Delay vs. Time



(b) Average Battery Occupancy vs. Time



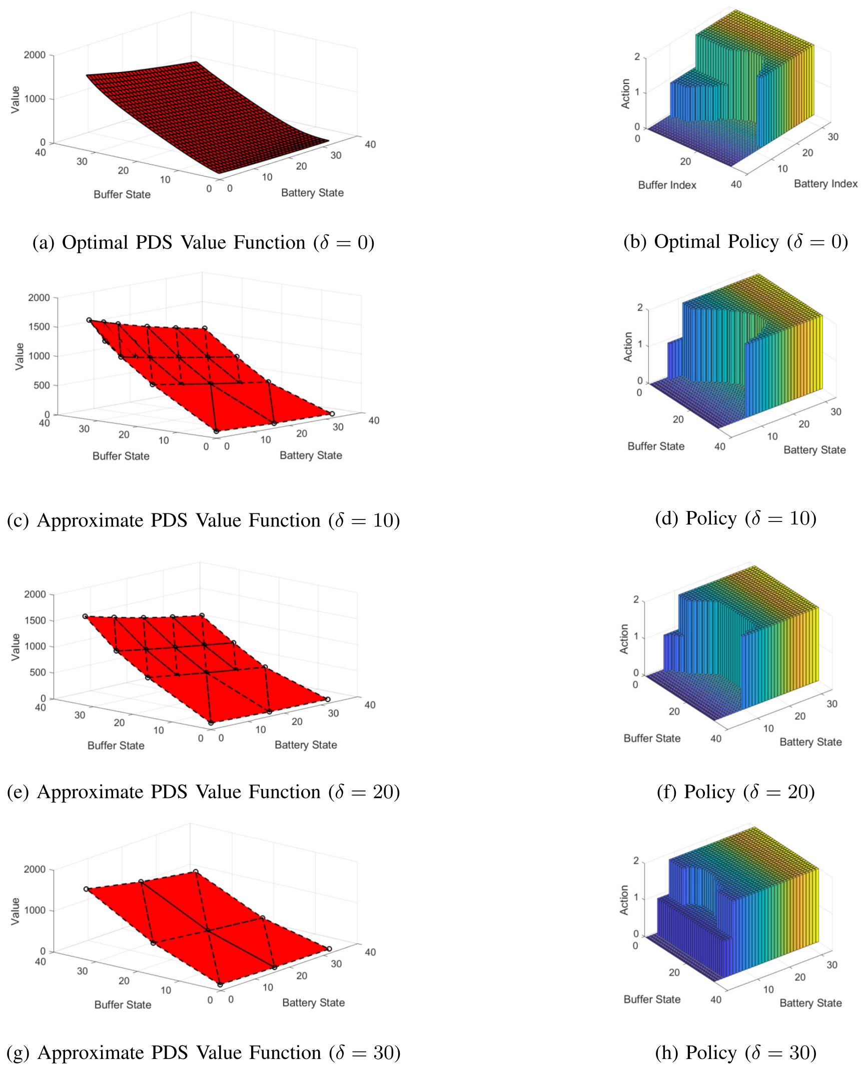
(c) Average Overflows vs. Time

Fig. 5. Performance comparison of the grid, PDS, and virtual experience learning algorithms.

Q-learning and Approximate SARSA perform worse than the other algorithms. This is due to the fact that: 1) they require action exploration [22], [35], so they frequently choose sub-optimal actions even if they have found the optimal action; and 2) they can only learn about one state-action pair in each time slot. Although PDS learning outperforms Q-learning and Approximate SARSA, it also takes an unacceptably long time to converge to the optimal solution because it can only learn about one PDS in each time slot. We observe that "Grid-10" achieves comparable delay performance to "VE-10" and "Optimal" in 10,000 and 50,000 time slots, respectively. Importantly, grid learning achieves this by updating 86% fewer states at a time compared to VE learning (at most 143 states for grid learning versus $(N_b + 1) \times (N_e + 1) = 1089$ for VE learning) and without any a priori knowledge about the channel, data

[6] $P^x(x) = \text{Bern}(p)$ denotes that x is a Bernoulli random variable that takes value 1 with probability p and value 0 with probability $1 - p$.

[7] Throughout the simulation results, we use "queuing delay" to refer to the queuing delay experienced by packets that are admitted into the queue. Please see Appendix A for more details.)



(a) Optimal PDS Value Function ($\delta = 0$)

(b) Optimal Policy ($\delta = 0$)

(c) Approximate PDS Value Function ($\delta = 10$)

(d) Policy ($\delta = 10$)

(e) Approximate PDS Value Function ($\delta = 20$)

(f) Policy ($\delta = 20$)

(g) Approximate PDS Value Function ($\delta = 30$)

(h) Policy ($\delta = 30$)

Fig. 6. PDS value functions and their associated policies for different error thresholds ($h = -3.28$ dB, $P^l(l) = \text{Bern}(0.2)$, and $P^{e_H}(e_H) = \text{Bern}(0.7)$).

arrival, and energy harvesting dynamics as is required to compute the optimal solution. Owing to this, a near-optimal transmission policy can be efficiently learned online on an EHS. Both "Grid-50" and "Grid-100" achieve slightly worse delay performance than "VE-10" after 50,000 time slots. Intuitively, grid learning performs better with more frequent updates.

Fig. 5 also reveals how the system evolves over time. Since the learning algorithms have no a priori knowledge of the dynamics, they operate with suboptimal policies until they gain sufficient experience by interacting with the environment. This leads to an initial surge in the queuing delay and buffer overflows as the EHS harvests energy from the environment, but has not yet learned how many packets to transmit. SARSA, Q-learning and PDS learning perform particularly poorly in this "cold start" phase because, unlike VE and grid learning, they have to actually experience large buffer states and packet overflows to learn how to avoid them.

C. *Effect of the Approximation Error Threshold*

In this section, we investigate the effect of the approximation error threshold δ on grid learning. All results in this section were taken after 50,000 time slot simulations with grid learning updates applied every $T = 100$ slots.

In Fig. 6, we compare several approximate PDS value functions ($\delta = 10, 20, 30$) against the optimal PDS value function ($\delta = 0$) in channel state $h = -3.28$ dB with data packet arrival distribution $P^l(l) = \text{Bern}(0.2)$ and energy packet arrival distribution $P^{e_H}(e_H) = \text{Bern}(0.7)$. We also compare their associated policies. In Fig. 6 a, we observe that the optimal PDS value function is non-decreasing and has increasing differences in the buffer state and is non-increasing and has increasing differences in the energy state (cf. Propositions 2 and 3). By design, relaxing the error tolerance leads to coarser piece-wise planar approximations of the PDS value function. For instance, at approximation error thresholds 0, 10, 20, and 30, the PDS value function is represented by 1089, 23, 18, and 9 states, respectively. Interestingly, we can also see that the policies in Fig. 6 become more aggressive as we relax the error threshold, i.e., they choose to transmit packets at lower and lower battery states.

Fig. 7 illustrates how the approximation errors (left-hand side of (29)) and error bounds (right-hand side of (29)) vary for different error thresholds and data packet arrival rates. We observe that the error bound becomes tighter with respect to the true approximation error as the arrival rate increases, and that it is always less than the error threshold δ. This demonstrates that the *grid learning* algorithm achieves both bounded and controllable

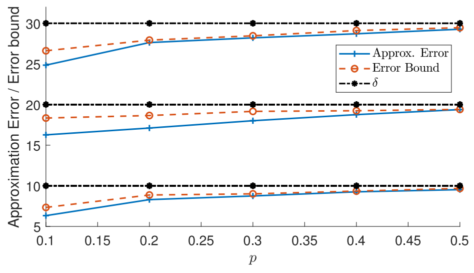
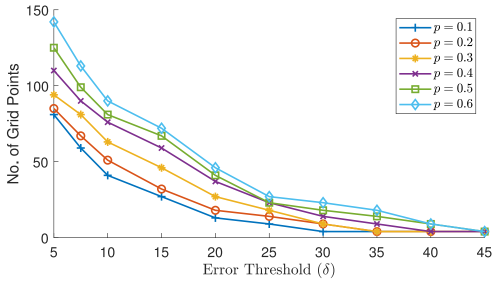
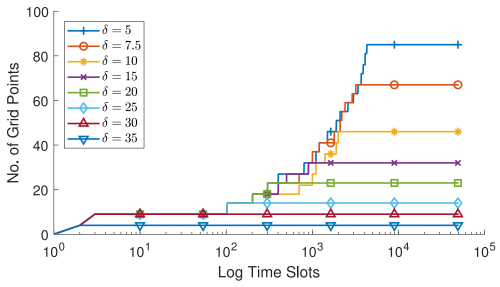


Fig. 7. Approximation error and Error bound for error thresholds $\delta = 10, 20, 30$ ($P^l(l) \sim Bern(p)$).



(a) Grid Points vs. Error Threshold δ, $P^l(l) = \text{Bern}(p)$.



(b) Grid Points vs. Time, $P^l(l) = \text{Bern}(0.2)$.

Fig. 8. Number of grid points for different error thresholds δ ($h = -3.28$ dB, and $P_{e_H} = \text{Bern}(0.7)$).

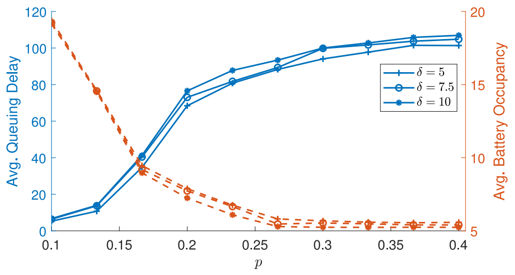


Fig. 9. Queuing Delay vs. Average Battery Occupancy ($P^l(l) = Bern(p)$ and $P_{e_H}(e_H) = \text{Bern}(0.7)$).

error, and validates our choice of the proposed quadtree-based adaptive piece-wise planar approximation.

Fig. 8(a) illustrates the number of grid points used to approximate the PDS value function versus the approximation error threshold δ for several data packet arrival rates. The measurements were taken from the approximate PDS value function in channel state $h = -3.28$ dB. These results further highlight that the number of grid points used in the PDS value function approximation decreases as the approximation error threshold increases. This intuitively follows from the fact that higher (resp. lower) error thresholds can be met by coarser (resp. finer) quadtree decompositions. We also observe that, for a fixed energy packet arrival rate, the number of grid points needed to meet a given error threshold roughly increases with the data packet arrival rate. This happens because the PDS value function's slope increases with the data packet arrival rate, which results in a larger approximation error at a fixed quadtree decomposition level (cf. Proposition 4). For instance, when $P^l(l) = \text{Bern}(0.6)$, the number of grid points needed to approximate the PDS value function within an error threshold of $\delta = 5$ is close to 150 points, which is almost twice the number of grid points needed to meet the same error threshold when $P^l(l) = \text{Bern}(0.1)$. This demonstrates that grid learning can adapt to the experienced dynamics.

Fig. 8(b) illustrates how the quadtree decomposition evolves over time to meet the approximation error threshold. Measurements were taken from the approximate PDS value function in channel state $h = -3.28$ dB. As before, the terminal number of grid points is lower for higher approximation error thresholds, δ. From the figure, we can see that the grid undergoes a lot of refinement in the first 4000 time slots to meet the error threshold. This manifests as a step-wise increase in the number of grid points every $T_{\text{grid}} = 100$ time slots. Note that, subdividing a leaf node can introduce 1-5 new grid points depending on the refinement level of the surrounding leaf nodes; therefore, the step sizes are variable over time.

Finally, Fig. 9 illustrates the average queuing delay and battery occupancy versus the data packet arrival rate at three different error thresholds. As expected, for a fixed energy packet arrival rate, the average queuing delay displays complementary behavior to the average battery occupancy. This is because, at low data arrival rates, the buffer can be kept small using a small fraction of the available energy. However, at high data arrival rates, more of the available energy is needed to keep the buffer occupancy from growing. In parallel, as the data arrival rate increases towards the channel's maximum service rate, the average buffer state increases. We also observe that tighter error thresholds yield better delay-energy trade-offs. For instance, $\delta = 5$ results in a lower average queuing delay while maintaining a higher average battery occupancy than $\delta = 10$. This can be explained by the fact that more accurate PDS value function approximations translate to better transmission scheduling policies.

VII. Conclusion

Foresighted decision making is required to minimize packet queuing delays on resource-constrained energy harvesting wireless sensors. In practice, however, the lack of a priori knowledge about the system's experienced dynamics presents a major challenge. Online RL represents a natural paradigm for overcoming this challenge, but generic RL algorithms are often unable to meet the stringent requirements of such systems in terms of memory, complexity, and convergence speed. Evidently, this requires carefully exploiting the structure of the problem at hand.

In this paper, we formulate the delay-sensitive energy harvesting scheduling problem as an MDP. We develop a low-complexity RL algorithm based on value function approximation, which exploits the structural properties of the optimal value function. The proposed algorithm allows us to learn an accurate approximation of the optimal value function online – with both bounded and controllable error – which in turn enables effective minimization of the packet queuing delay. We demonstrate that the proposed algorithm achieves near optimal performance. Moreover, competitive performance is demonstrated relative to a state-of-the-art learning algorithm, at potentially orders of magnitude lower computational complexity. Finally, our framework enables considerable performance gains over the widely used Q-learning algorithm.

Appendix A
Buffer Cost Definition

We explain here that our choice of buffer cost allows us to minimize the expected delay of a packet admitted into the buffer, while meeting an implicitly defined constraint on the packet overflow probability.

Recall that we aim to find the policy $\pi : \mathcal{S} \to \mathcal{A}$ that is the solution to the following optimization problem:

$$\min_{\pi} \mathbb{E}\left[\sum_{n=0}^{\infty} (\gamma)^n c(s^n, \pi(s^n))\right], \tag{32}$$

where

$$c(s^n, a^n) = b^n + \eta \mathbb{E}_{f,l}\{\max(b^n - f + l - N_b, 0)\}. \tag{33}$$

Note that (32) and (33) are simply restatements of (10) and (14), respectively. The above problem can be interpreted as an unconstrained reformulation of the following constrained MDP (CMDP [38]):

$$\min_{\pi} \mathbb{E}\left[\sum_{n=0}^{\infty} (\gamma)^n b^n\right], \tag{34}$$

$$\text{s.t.: } \mathbb{E}\left[\sum_{n=0}^{\infty} (\gamma)^n \mathbb{E}_{f,l}\{\max(b^n - f + l - N_b, 0)\}\right] \leq \frac{1}{1-\gamma}\bar{P}_O, \tag{35}$$

where the objective is to minimize the discounted queue backlog subject to the packet overflow probability constraint $\bar{P}_O$. Note that the $\frac{1}{1-\gamma}$ term is necessary to convert the packet overflow probability constraint $\bar{P}_O$ to a *discounted* packet overflow probability constraint, since the left-hand side of (35) represents the discounted packet overflow probability under policy π. Specifically, (32) reformulates the constrained MDP defined in (34) and (35) as an unconstrained MDP via a Lagrangian cost function defined in (33), with Lagrange multiplier η associated with the packet overflow probability constraint. The optimal Lagrange multiplier η that meets this constraint can be learned online using stochastic subgradient methods [23], [31]; however, this lies beyond the scope of this paper. Instead, we choose to directly select η.[8]

For an infinite data buffer, Little's law [39] tells us that the expected queue backlog $\bar{Q}$ is proportional to the expected queuing delay $\bar{D}$ if the buffer is stable, i.e.,

$$\bar{Q} = \lambda \bar{D},$$

where λ is the expected arrival rate. Note that Little's law still holds if we replace the expected queue backlog and the expected queuing delay with their discounted equivalents. In the finite data buffer scenario where overflows are inescapable, Little's law can be applied to calculate the expected delay experienced by packets that are *admitted* into the buffer using the *effective arrival rate*, i.e., $(1 - P_O)\lambda$, where P_O is the packet overflow probability.

The optimal policy that minimizes (32) will exactly meet the implicit packet overflow probability constraint $\bar{P}_O$ induced by the choice of η. Thus, by solving (32), we are effectively minimizing the expected queue backlog $\bar{Q} = (1 - P_O)\lambda \bar{D}$, where the packet overflow probability $P_O = \bar{P}_O$ does not depend on the decision policy. Given the latter, it hence follows that our choice of buffer cost allows us to minimize the expected delay of a packet admitted into the buffer (i.e., $\bar{D}$), while meeting an implicitly defined constraint on the packet overflow probability (i.e., $\bar{P}_O$, which depends on η).

Appendix B
PDS Value Function Bellman Equations

Using the PDS, we can factor the transition probabilities into known and unknown components, where the known component accounts for the transition from the current state to the PDS ($s \to \tilde{s}$) and the unknown component accounts for the transition from the PDS to the next state ($\tilde{s} \to s'$) [23]:

$$P(s'|s, a) = \sum_{\tilde{s} \in \mathcal{S}} p_u(s'|\tilde{s}) p_k(\tilde{s}|s, a), \tag{36}$$

where subscripts k and u denote the known and unknown components, respectively. We factor the cost function similarly:

$$c(s, a) = c_k(s, a) + \sum_{\tilde{s} \in \mathcal{S}} p_k(\tilde{s}|s, a) c_u(\tilde{s}). \tag{37}$$

In our problem, the known and unknown costs and transition probabilities are defined as:

$$c_k(s, a) = b, \tag{38}$$

$$c_u(\tilde{s}) = \eta \sum_{l \in \mathcal{L}} P^l(l) \max(\tilde{b} + l - N_b, 0), \tag{39}$$

$$P_k(\tilde{s}|s, a) = P^f(b - \tilde{b}|a) \mathbb{I}_{\{\tilde{e} = e - e_{TX}(h,a)\}} \mathbb{I}_{\{\tilde{h} = h\}}, \tag{40}$$

$$P_u(s'|\tilde{s}) = \sum_{l \in \mathcal{L}} \sum_{e_H \in \mathcal{E}} \mathbb{I}_{\{b' = \min(\tilde{b} + l, N_b)\}} \mathbb{I}_{\{e' = \min(\tilde{e} + e_H, N_e)\}} P_l(l) P_{e_H}(e_H) P_h(h'|h) \tag{41}$$

where $\mathbb{I}_{\{\cdot\}}$ is an indicator function that is set to 1 when $\{\cdot\}$ is true and is set to 0 otherwise; $\tilde{s}$ denotes the PDS tuple $(b - f, e - e_{TX}(h, a), h)$; and s' denotes the next state tuple $(\tilde{b} + l, \tilde{e} + e_H, h')$.

We can now rewrite equation (18) using the known and unknown costs and transition probabilities as:

$$\begin{aligned}\tilde{V}^*(\tilde{s}) &= c_u(\tilde{s}) + \gamma \sum_{s' \in \mathcal{S}} P_u(s'|\tilde{s}) V^*(s') \\ &= \eta \sum_{l \in \mathcal{L}} P^l(l) \max(\tilde{b} + l - N_b, 0) \\ &\quad + \gamma \sum_{l, e_H, h'} P^l(l) P^{e_H}(e_H) P^h(h'|h) \\ &\quad \times V^*([\min(\widetilde{b} + l, N_b), \min(\widetilde{e} + l, N_e), h']),\end{aligned} \tag{42}$$

[8]We select η according to a heuristic proposed in our prior work [23] Specifically, we set $\eta = 1/(1 - \gamma)$, which penalizes each dropped packet by the discounted cost that the packet would have incurred if it were to sit in the buffer forever (i.e., the worst-case long-run holding cost for a packet). This ensures that the policy will always prefer to leave space in the buffer to admit packets, if possible, rather than to drop them.

where the second equality follows from the definitions of $c_u(\tilde{s})$ and $P_u(s'|\tilde{s})$ in (39) and (41), respectively. We can similarly rewrite Eq. (19) as,

$$V^*(s) = \min_{a\in\mathcal{A}(s)} \left\{ c_k(s,a) + \sum_{\tilde{s}\in\mathcal{S}} P_k(\tilde{s}|s,a)\tilde{V}^*(\tilde{s}) \right\}$$
$$= \min_{a\in\mathcal{A}(b,e,h)} \left\{ b + \sum_{f=0}^{a} P^f(f|a)\tilde{V}^*([b-f, e-e_{TX}(h,a), h]) \right\} \tag{43}$$

where the second equality follows from the definitions of $c_k(s,a)$ and $P_k(\tilde{s}|s,a)$ in (38) and (40), respectively.

Substituting (42) into (44), we get,

$$V^*(s) = \min_{a\in\mathcal{A}(s)} \left\{ c_k(s,a) + \sum_{\tilde{s}\in\mathcal{S}} P_k(\tilde{s}|s,a)\tilde{V}^*(\tilde{s}) \right\}.$$
$$= \min_{a\in\mathcal{A}(s)} \left\{ c_k(s,a) + \sum_{\tilde{s}\in\mathcal{S}} P_k(\tilde{s}|s,a)[c_u(\tilde{s}) + \gamma \sum_{s'\in\mathcal{S}} P_u(s'|\tilde{s})V^*(s')] \right\}$$
$$= \min_{a\in\mathcal{A}(s)} \left\{ c_k(s,a) + \sum_{\tilde{s}\in\mathcal{S}} P_k(\tilde{s}|s,a)c_u(\tilde{s}) + \gamma \sum_{s'\in\mathcal{S}}\sum_{\tilde{s}\in\mathcal{S}} P_u(s'|\tilde{s})P_k(\tilde{s}|s,a)V^*(s') \right\}$$
$$= \min_{a\in\mathcal{A}(s)} \left\{ c(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)V^*(s') \right\},$$

where the final equality follows from (36) and (37). Thus, the Bellman equations defined in (19) and (18) natural follow from the definition of the PDS, and they are equivalent to the standard Bellman equation defined in (11).

Appendix C
Proof of Proposition 1

The first condition in Definition 1 is satisfied by assumption. We only need to show that the PDS Learning algorithm satisfies the second and third conditions.

To simplify the proof, we use the notation introduced in Appendix B. For brevity, we will omit the conditioning variables in (40) and (41), i.e., we will write $P_k(\tilde{s})$ and $P_u(s')$ instead of $P_k(\tilde{s}|s,a)$ and $P_u(s'|\tilde{s})$, respectively. Plugging (43) into (42), we can define a mapping H_{PDS} that maps a $\tilde{V}$-vector to a new $\tilde{V}$-vector $H_{PDS}\tilde{V}$ as

$$(H_{PDS}\tilde{V})(\tilde{s}) = c_u(\tilde{s}) + \gamma \sum_{s'\in\mathcal{S}} P_u(s') \min_{a\in\mathcal{A}} \left\{ c_k(s',a) + \sum_{\tilde{s}'\in\mathcal{S}} P_k(\tilde{s}')\tilde{V}(\tilde{s}') \right\}, \tag{44}$$

where $\tilde{s}$, s', and $\tilde{s}'$ denote the current PDS, next state, and next PDS, respectively. Now, we can rewrite the PDS learning update in (22) using H_{PDS}:

$$\tilde{V}^{n+1}(\tilde{s}^n) = (1-\beta^n)\tilde{V}^n(\tilde{s}^n) + \beta^n \times \left[(H_{PDS}\tilde{V}^n)(\tilde{s}^n) + w^n(\tilde{s}^n) \right],$$
$$w^n(\tilde{s}^n) = \eta \max(\tilde{b}^n + l^n - N_b, 0) + \gamma V^n(s^{n+1}) - \left[c_u(\tilde{s}^n) + \gamma \sum_{s'\in\mathcal{S}} P_u(s')V^n(s') \right].$$

For any history F^n, it is easy to show that $E[w^n(\tilde{s}^n)|F^n] = 0$ and $|w^n(\tilde{s}^n)| \le V_{\max}$, where $V_{\max} = \max\{c(s,a)\}/(1-\gamma)$. We need to show that H_{PDS} satisfies the contraction property:

$$\left|(H_{PDS}\tilde{V})(\tilde{s}) - \tilde{V}^*(\tilde{s})\right|$$
$$= \gamma \sum_{s'\in\mathcal{S}} P_u(s') \left|V(s') - V^*(s')\right|$$
$$= \gamma \sum_{s'\in\mathcal{S}} P_u(s') \left| \min_{a\in\mathcal{A}} \left\{ c_k(s',a) + \sum_{\tilde{s}'\in\mathcal{S}} P_k(\tilde{s}')\tilde{V}(\tilde{s}') \right\} - \min_{a\in\mathcal{A}} \left\{ c_k(s',a) + \sum_{\tilde{s}'\in\mathcal{S}} P_k(\tilde{s}')\tilde{V}^*(\tilde{s}') \right\} \right|$$
$$\le \gamma \sum_{s'\in\mathcal{S}} P_u(s') \max_{a\in\mathcal{A}} \left| \sum_{\tilde{s}'\in\mathcal{S}} P_k(\tilde{s}')\tilde{V}(\tilde{s}') - \sum_{\tilde{s}'\in\mathcal{S}} P_k(\tilde{s}')\tilde{V}^*(\tilde{s}') \right|$$
$$= \gamma \sum_{s'\in\mathcal{S}} P_u(s') \max_{a\in\mathcal{A}} \sum_{\tilde{s}'\in\mathcal{S}} P_k(\tilde{s}') \left| \left(\tilde{V}(\tilde{s}') - \tilde{V}^*(\tilde{s}') \right) \right|$$
$$\le \gamma \sum_{s'\in\mathcal{S}} P_u(s') \max_{a\in\mathcal{A}} \sum_{\tilde{s}'\in\mathcal{S}} P_k(\tilde{s}') ||\tilde{V} - \tilde{V}^*||$$
$$= \gamma ||\tilde{V} - \tilde{V}^*||,$$

where the first and second equalities follow by applying the definition of $(H_{PDS}\tilde{V})(\tilde{s})$ in (44); the first inequality follows from the fact that the difference of minimums is less than the maximum of differences; the third equality follows by rearranging terms; the final inequality follows by definition of the L_∞ norm; and the last equality follows from the fact that $||\tilde{V} - \tilde{V}^*||$ does not depend on the summation variables s' and $\tilde{s}'$, and $P_u(s')$ and $P_k(\tilde{s}')$ sum to 1. □

References

[1] J. Chakareski, "Uplink scheduling of visual sensors: When view popularity matters," *IEEE Trans. Commun.*, vol. 2, no. 63, pp. 510–519, Feb. 2015.

[2] J. Chakareski, "Informative state-based video communication," *IEEE Trans. Image Process.*, vol. 22, no. 6, pp. 2115–2127, Jun. 2013.

[3] A. Seyedi and B. Sikdar, "Energy efficient transmission strategies for body sensor networks with energy harvesting," *IEEE Trans. Commun.*, vol. 58, no. 7, pp. 2116–2126, Jul. 2010.

[4] J. Chakareski, "UAV-IoT for next generation virtual reality," *IEEE Trans. Image Process.*, vol. 28, no. 12, pp. 5977–5990, Dec. 2019.

[5] J. Chakareski, "VR/AR immersive communication: Caching, edge computing, and transmission trade-offs," in *Proc. ACM Workshop Virtual Reality Augmented Reality Netw.*, 2017, pp. 36–41.

[6] X. Corbillon, A. Devlic, G. Simon, and J. Chakareski, "Viewport-adaptive navigable 360-degree video delivery," in *Proc. IEEE Int'l Conf. Commun.*, Paris, France, May 2017, Art. no. 17065493.

[7] D. Zordan, T. Melodia, and M. Rossi, "On the design of temporal compression strategies for energy harvesting sensor networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 2, pp. 1336–1352, Feb. 2016.

[8] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava,“Power management in energy harvesting sensor networks,” *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 4, p. 32, 2007.

[9] R. J. Vullers, R. V. Schaijk, H. J. Visser, J. Penders, and C. V. Hoof, “Energy harvesting for autonomous wireless sensor networks,” *IEEE Solid-State Circuits Mag.*, vol. 2, no. 2, pp. 29–38, 2010.

[10] B. Gurakan and S. Ulukus, “Energy harvesting multiple access channel with data arrivals,” in *Proc. IEEE Global Commun. Conf.*, 2015, pp. 1–6.

[11] X. Lu, P. Wang, D. Niyato, and E. Hossain, “Dynamic spectrum access in cognitive radio networks with RF energy harvesting,” *IEEE Wireless Commun.*, vol. 21, no. 3, pp. 102–110, Jun. 2014.

[12] V. Sharma, U. Mukherji, V. Joseph, and S. Gupta, “Optimal energy management policies for energy harvesting sensor nodes,” *IEEE Trans. Wireless Commun.*, vol. 9, no. 4, Apr. 2010, pp. 1326–1336.

[13] D. Gunduz, K. Stamatiou, N. Michelusi, and M. Zorzi, “Designing intelligent energy harvesting communication systems,” *IEEE Commun. Mag.*, vol. 52, no. 1, pp. 210–216, Jan. 2014.

[14] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2014.

[15] O. Ozel, K. Tutuncuoglu, J. Yang, S. Ulukus, and A. Yener, “Transmission with energy harvesting nodes in fading wireless channels: Optimal policies,” *IEEE J. Sel. Areas Commun.*, vol. 29, no. 8, pp. 1732–1743, Sep. 2011.

[16] C. Ho and R. Zhang, “Optimal energy allocation for wireless communications with energy harvesting constraints,” *IEEE Trans. Signal Process.*, vol. 60, no. 9, pp. 4808–4818, Sep. 2012.

[17] J. Yang and S. Ulukus, “Optimal packet scheduling in a multiple access channel with energy harvesting transmitters,” *J. Commun. Netw.*, vol. 14, no. 2, pp. 140–150, 2012.

[18] J. Yang and S. Ulukus, “Optimal packet scheduling in an energy harvesting communication system,” *IEEE Trans. Commun.*, vol. 60, no. 1, pp. 220–230, Jan. 2012.

[19] N. Michelusi, K. Stamatiou, and M. Zorzi, “On optimal transmission policies for energy harvesting devices,” in *Proc. Inf. Theory Appl. Workshop*, IEEE, 2012, pp. 249–254.

[20] A. Aprem, C. R. Murthy, and N. B. Mehta, “Transmit power control policies for energy harvesting sensors with retransmissions,” *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 5, pp. 895–906, Oct. 2013.

[21] C. K. Ho and R. Zhang, “Optimal energy allocation for wireless communications powered by energy harvesters,” in *Proc. IEEE Int. Symp. Inf. Theory*, 2010, pp. 2368–2372.

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

[23] N. Mastronarde and M. van der Schaar, “Joint physical-layer and system-level power management for delay-sensitive wireless communications,” *IEEE Trans. Mobile Comput.*, vol. 12, no. 4, pp. 694–709, Apr. 2013.

[24] P. Blasco, D. Gunduz, and M. Dohler, “A learning theoretic approach to energy harvesting communication system optimization,” *IEEE Trans. Wireless Commun.*, vol. 12, no. 4, pp. 1872–1882, Apr. 2013.

[25] C. J. Watkins and P. Dayan, “Q-learning,” *Mach. Learn.*, vol. 8, no. 3-4, pp. 279–292, 1992.

[26] A. Ortiz, H. Al-Shatri, X. Li, T. Weber, and A. Klein, “Reinforcement learning for energy harvesting point-to-point communications,” in *Proc. IEEE Int. Conf. Commun.*, Kuala Lumpur, Malaysia, May 2016, pp. 1–6.

[27] Y. Xiao, Z. Han, D. Niyato, and C. Yuen, “Bayesian reinforcement learning for energy harvesting communication systems with uncertainty,” in *IEEE Int. Conf. Commun.*, 2015, pp. 5398–5403.

[28] N. Mastronarde and M. van der Schaar, “Fast reinforcement learning for energy-efficient wireless communication,” *IEEE Trans. Signal Process.*, vol. 59, no. 12, pp. 6262–6266, Dec. 2011.

[29] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, vol. 703, New York, NY, USA: Wiley, 2007.

[30] D. V. Djonin and V. Krishnamurthy, “Mimo transmission control in fading channels a constrained markov decision process formulation with monotone randomized policies,” *IEEE Trans. Signal Process.*, vol. 55, no. 10, pp. 5069–5083, Oct. 2007.

[31] N. Salodkar, A. Bhorkar, A. Karandikar, and V. Borkar, “An on-line learning algorithm for energy efficient delay constrained scheduling over a fading channel,” *IEEE J. Sel. Areas Commun.*, vol. 26, no. 4, pp. 732–742, May 2008.

[32] Q. Zhang and S. A. Kassam, “Finite-state Markov model for Rayleigh fading channels,” *IEEE Trans. Commun.*, vol. 47, no. 11, pp. 1688–1692, Nov. 1999.

[33] M. H. Ngo and V. Krishnamurthy, “Monotonicity of constrained optimal transmission policies in correlated fading channels with ARQ,” *IEEE Trans. Signal Process.*, vol. 58, no. 1, pp. 438–451, Jan. 2010.

[34] A. Goldsmith, *Wireless Communications*. New York, NY, USA: Cambridge Univ. Press, 2005.

[35] L. P. Kaelbling, L. M. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.

[36] D. P. Bertsekas and J. N. Tsitsiklis, “Neuro-dynamic programming: An overview,” in *Proc. 34th IEEE Conf. Decis. Control*, vol. 1, 1995, pp. 560–564.

[37] N. Sharma, N. Mastronarde, and J. Chakareski, “Delay-sensitive energy harvesting wireless sensors: Optimal scheduling, structural properties, and approximation analysis,” *IEEE Trans. Commun.*, 2019, to be published.

[38] E. Altman, *Constrained Markov Decision Processes*, vol. 7. Boca Raton, FL, USA: CRC Press, 1999.

[39] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data Networks*, vol. 2. Englewood Cliffs, NJ: Prentice-Hall 1987.

Nikhilesh Sharma received the B.Tech. degree in electronics and communication engineering from the National Institute of Technology Srinagar, Srinagar, India, in 2014, and the M.S. degree in electrical engineering in 2017 from the University at Buffalo, Buffalo, NY, USA, where he is currently working toward the Ph.D. degree. His research interests include reinforcement learning, deep learning, Markov decision processes, resource allocation in wireless networks and systems, and 4 G/5 G networks.

Nicholas Mastronarde (Senior Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the University of California, Davis, CA, USA, in 2005 and 2006, respectively, and the Ph.D. degree in electrical engineering from the University of California, Los Angeles, CA, USA, in 2011. He is currently an Associate Professor with the Department of Electrical Engineering, University at Buffalo, Buffalo, NY, USA. His research interests include reinforcement learning, Markov decision processes, resource allocation and scheduling in wireless networks and systems, UAV networks, and 4 G/5 G networks.

Jacob Chakareski (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Rice University, Houston, TX, USA and Stanford University, Stanford, CA, USA. He held research appointments with Microsoft, HP Labs, and EPFL, and sat on the advisory board of Frame, Inc. He is currently an Associate Professor with the Ying Wu College of Computing, NJIT, Newark, NJ, USA, where he leads the Laboratory for VR/AR Immersive Communication (LION). His research interests span networked virtual and augmented reality, UAV IoT sensing and networking, real-time reinforcement learning, 5G wireless edge computing/caching, ubiquitous immersive communication, and societal applications. His research was supported by National Science Foundation (NSF), AFOSR, Adobe, Tencent Research, NVIDIA, and Microsoft. He is the organizer of the first NSF Visioning Workshop on networked VR/AR communications. He was the recipient of the Adobe Data Science Faculty Research Award in 2017 and 2018, the Swiss NSF Career Award Ambizione in 2009, the AFOSR Faculty Fellowship in 2016 and 2017, and the Best/Fast-Track Paper Awards at IEEE ICC 2017 and IEEE Globecom 2016.