# Improving Data-Driven Reinforcement Learning in Wireless IoT Systems Using Domain Knowledge

Nicholas Mastronarde, Nikhilesh Sharma, and Jacob Chakareski

## Abstract

Reinforcement learning (RL) algorithms are purely data-driven and do not leverage any domain knowledge about the nature of the available actions, the system's state transition dynamics, and its cost/reward function. This severely penalizes their ability to meet critical requirements of emerging wireless applications, due to the inefficiency with which these algorithms learn from their interactions with the environment. In this article, we describe how data-driven RL algorithms can be improved by systematically integrating basic system models into the learning process. Our proposed approach uses real-time data in conjunction with knowledge about the underlying communication system to achieve orders of magnitude improvement in key performance metrics, such as convergence speed and compute/memory complexity, relative to well-established RL benchmarks.

## Introduction

Wirelessly networked Internet of Things (IoT) devices increasingly need to operate in unknown dynamic environments comprising latency-sensitive data sources and wireless channels of a priori unknown statistics. This challenge is encountered in emerging applications such as personalized healthcare, remote robot navigation, telemetry, and mobile virtual and augmented reality (VR/AR). Paramount to their effective operation will be the ability to communicate the captured data in real time, despite the IoT devices' limited capabilities and their unknown operating environments. This will require advances in online learning and decision making to effectively leverage the IoT devices' limited resources while meeting the application's critical service constraints.

Optimal decision making in unknown environments, where decision policies need to be learned online, is a natural application domain of reinforcement learning (RL). Here, an agent interacts with an unknown environment and updates its *value function* and *policy* based on its experience, where the value function describes how good it is to be in each state and the policy describes what action to take in each state. As the agent acquires experience, its value function and policy converge to their optimal forms. RL has been used to solve a variety of important problems in wireless IoT systems, such as compression and transmission rate control [1, 2], duty cycle optimization [3], multi-access control [4], autonomous ad hoc network formation [5], fresh data collection [6], and edge computing [7]. However, traditional RL algorithms — including model-free and model-based, on-policy and off-policy, and value-based and policy gradient methods [8] — are purely data-driven and operate without exploiting any domain knowledge about the nature of the system's states and actions, its state transition dynamics, or the performance metrics that need to be optimized. This allows them to solve a wide variety of problems, but yields algorithms that either:

1. Converge too slowly to be useful in wireless IoT systems, where fast adaptation to the experienced channel, information source, and network dynamics is essential (e.g., model-free solutions, such as Q-learning, SARSA, and policy gradient and actor-critic methods)
2. Are too computationally and memory intensive to implement on resource constrained IoT devices (e.g., model-based solutions, such as adaptive real-time dynamic programming [ARTDP], Dyna, and Dyna-Q)

To bridge this gap, we present a systematic approach to improve data-driven RL algorithms by integrating domain knowledge — in the form of basic system models — into the learning process. This approach comprises three components:

- **Post-decision states** (also known as afterstates [8]) allow us to factor the system's cost/reward function and state transition probability function into *known* and *unknown* components. This accelerates learning but increases computational complexity.
- **Virtual experience**, a novel technique that we developed, allows us to extrapolate the observed experience in one system state to other system states. This further accelerates learning, but also further increases computational complexity.
- **Value function approximation** allows us to reap the benefits of virtual experience while mitigating its computational complexity. Moreover, under certain conditions, the resulting approximation error is provably bounded.

The authors describe how data-driven RL algorithms can be improved by systematically integrating basic system models into the learning process. Their proposed approach uses real-time data in conjunction with knowledge about the underlying communication system to achieve orders of magnitude improvement in key performance metrics, such as convergence speed and compute/memory complexity, relative to well-established RL benchmarks.

The work of N. Mastronarde and N. Sharma was supported by NSF Award ECCS-1711335. The work of J. Chakareski was supported by NSF Awards CCF-1528030, ECCS-1711592, CNS-1836909, CNS-1821875, and CNS-1836909.

*Nicholas Mastronarde and Nikhilesh Sharma are with the University of Buffalo; Jacob Chakareski is with the New Jersey Institute of Technology.*

Digital Object Identifier: 10.1109/MCOM.111.2000949

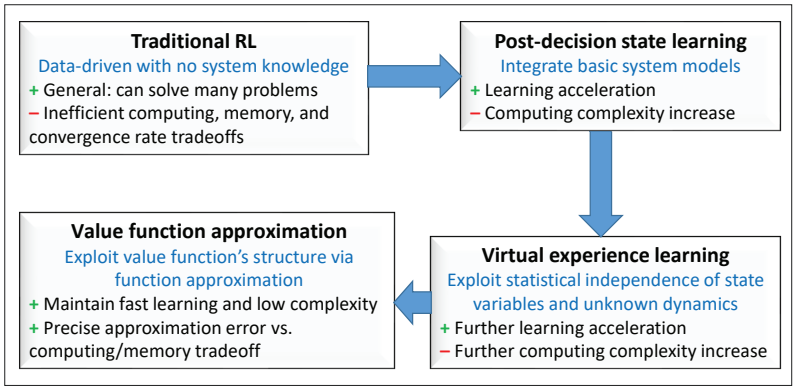


FIGURE 1. Improving data-driven RL using domain knowledge.

*Each of these components require us to leverage domain knowledge to:*

1. *Verify that a system satisfies the conditions under which they are applicable*
2. *Identify how to correctly implement them for a specific system*

In this article, we emphasize problems with finite state and action spaces that can be solved with *tabular* RL methods, where the value functions and policies can be represented with tables. However, the proposed approach is general and can also be applied to problems with continuous or infinite state and action spaces. Deep RL, which uses deep neural networks to approximate the value function and/or policy, has emerged as a promising approach to solve such problems, particularly when they have high-dimensional state and action spaces. However, deep RL algorithms are purely data-driven just like conventional RL algorithms. As such, we also discuss the prospective application of our framework in deep RL. Additionally, we assume that the RL algorithm operates online at runtime. This is in contrast to so-called robust RL [9] in which the RL model is "trained" in a different environment than it is "tested." Importantly, recent advances in both deep RL and robust RL are orthogonal to the advances we focus on herein. Lastly, note that our approach can be applied in any wireless system under the appropriate conditions, but we focus on wireless IoT systems due to the constraints imposed in this environment, as highlighted earlier.

Our systematic framework, its principal components, and their characteristics are summarized in Fig. 1. In the remainder of this article, we provide tutorial-level coverage of these concepts. We begin with an example system model that we will use to illustrate the proposed approach, and provide further background on the limitations of existing solutions. We proceed with an overview of the three principal components of our framework, and discuss their prospective extension to multi-agent RL and deep RL. Finally, we review related work and conclude the article.

## Example System Model and Background

We introduce in Fig. 2 a simple example system model — focusing on energy-efficient point-to-point scheduling of delay-sensitive data over a fading channel — to illustrate our proposed approach. We selected this simple model because it shares key characteristics with more complex IoT systems. We assume that the system operates in discrete time. In each time step, the transmission scheduler observes the buffer state and channel state, and then decides the number of packets to transmit. At the end of each step, new packets arrive into the transmission buffer from the information source. The buffer state evolves over time, driven by the packet transmissions and arrivals, and the channel state evolves according to a finite-state Markov chain. The data arrival probability distribution and channel state transition probabilities are assumed to be unknown.

We introduce a transmission cost defined as the power required to transmit different numbers of packets in different channel states, while meeting a target bit error rate. Similarly, we introduce a buffer cost to penalize buffer delays and overflows, which depend on the buffer state, packet transmissions, and packet arrivals. The transmission scheduler decides how many packets to transmit in each state to minimize the average transmission power subject to an average buffer cost constraint.

### Constrained Markov Decision Process Formulation

The example problem described above can be formulated as a constrained Markov decision process (MDP) [10], where the system's *state* comprises its buffer and channel states, and the *action* denotes the number of packets to transmit in the time step. The state evolves as a controlled Markov chain with *transition probabilities* that depend on the current state, selected action, data arrival probability distribution, and channel transition probabilities. The objective is to take the action in each step that minimizes the average *transmission cost* subject to an average *buffer cost* constraint.

The problem can be reformulated as an unconstrained optimization using the Lagrange multiplier (LM) method [10]. This is done by introducing a Lagrangian cost function defined as the sum of the transmission cost and the buffer cost, where the buffer cost is multiplied by an LM. For a given LM, the optimal solution satisfies a Bellman equation featuring the optimal *value function*, which indicates how good it is to be in each state when following the optimal policy. The related optimal *action-value function* indicates how good it is to take an arbitrary action in each state and then follow the optimal policy thereafter. The optimal value function can be determined from the optimal action-value function by minimizing it over the actions. The optimal *policy* captures the optimal action to take in each state.

### Limitations of Existing Approaches

If the transmission costs, buffer costs, and state transition probabilities are known, the optimal value function that satisfies the Bellman equation can be computed numerically using dynamic programming (e.g., value iteration [8]). Moreover, the optimal LM that satisfies the buffer cost constraint can be computed using the subgradient method [11]. In practice, however, the cost function and state transition probabilities are (partially) unknown, as the data arrival probability distribution and channel transition probabilities are unknown. Hence, the optimal value function and policy cannot be computed using dynamic programming; instead, they must be learned online, based on experience. This can be accomplished in one of two ways [12]:

- **Model-free:** Learn a value function or policy without learning a model.
- **Model-based:** Learn a model for the costs and transition probabilities, and then derive a value function or policy.

In parallel, the optimal LM can be learned using the stochastic subgradient method [11]. *Importantly, both model-free and model-based approaches are purely data-driven and assume no a priori information about the costs and transition probabilities.*

As an instance of the model-free approach, Q-learning updates an estimate of the action-value function in each step based on its observed *experience tuple*, comprising the current state, selected action, observed cost, and next state, using a time-varying step-size parameter. These iterations converge to the optimal action-value function with probability 1 if the step-size is defined appropriately. As an instance of the model-based approach, ARTDP estimates both the cost and transition probability functions, and then applies dynamic programming backups to update the value function using these estimates.

Unfortunately, as noted in the introduction, neither approach is well suited for real-time learning in IoT systems: model-free approaches have low computational complexity, but converge too slowly; and while model-based approaches often converge more quickly, they are too complex.

## Improving Data-Driven RL Using Domain Knowledge

In this section, we explore the sequential and synergistic application of post-decision states, virtual experience, and value function approximation to overcome the aforementioned limitations of data-driven RL algorithms.

### Post-Decision States and Virtual Experience

First, we generalize the concept of a post-decision state (PDS), which captures the system state after the known effects of an action, but before the unknown dynamics take place. Conceptually, a PDS can be viewed as an intermediate state in the transition from the current state to the next state, which we can model based on our knowledge of how the actions affect the state transition in a system. On the left side of Fig. 3, we illustrate the PDS concept in the context of the example system in Fig. 2. Here, we define the post-decision buffer state as the difference between the current buffer state and the number of transmitted packets, such that it represents an intermediate buffer state *after* packets are transmitted, but *before* new packets arrive. The next buffer state can then be calculated by adding the new packet arrivals to the post-decision buffer state, while accounting for any buffer overflows. Meanwhile, the post-decision channel state is the same as the channel state because we do not know the channel model. At first inspection, PDSs appear to be a simple mathematical trick; however, they have rather profound implications.

As noted in the introduction, the PDS concept exists in prior literature (e.g,. [8, 11]). However, this literature assumes that the PDS is a known *deterministic* function of the current state and action and that the cost function is completely *known*. This definition limits the concept's applicability: for example, it cannot be used to model packet losses from a known distribution (because the post-decision buffer state depends on the number of successfully transmitted packets, which is non-deterministic) or to model finite buffers (because the expected overflow cost depends on the arrival distribution, which is unknown). In contrast, our generalized PDS can be a known *non-deterministic* function of the current state and action, and allows the cost function to contain *known* and *unknown* components.

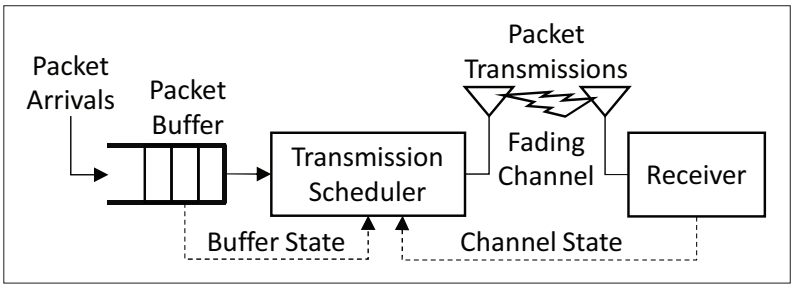


FIGURE 2. Example system model.

A generalized PDS can be introduced if the system's cost/reward and state transition probability functions can be factorized into known and unknown components. This fills the gap between MDPs and purely data-driven RL: If everything is known, the problem reduces to an MDP that can be solved using dynamic programming; if everything is unknown, it must be solved with a purely data-driven RL algorithm.

Just as we formulated a value function over the conventional states, we can formulate a *PDS value function* over the PDSs. Note that these two value functions are closely related: the value function depends on the expectation of the PDS value function with respect to (w.r.t.) the known transition probabilities; and the PDS value function depends on the expectation of the value function w.r.t. the unknown transition probabilities. In our example system model, the known transition from the current state to the PDS is deterministic (Fig. 3), so the value function simply depends on the PDS value function evaluated at the observed PDS. On the other hand, the unknown transition from the PDS to the next state depends on the unknown packet arrival and channel transition probabilities (Fig. 3).

Despite the close relationship between the two value functions, working with the PDS value function has several advantages. First, only the unknown information in the transition from the PDS to the next state needs to be learned. Second, by updating the value of one PDS, we learn about all state-action pairs that can precede it (Fig. 3). Third, in RL, there is a trade-off between *exploiting* actions that currently have the best estimated value and *exploring* other actions that might be better. However, in the special case that the unknown transition probabilities are independent of the action, exploration is not needed. For instance, in the example system in Fig. 2, the unknown transition probabilities depend on the packet arrival distribution and the channel

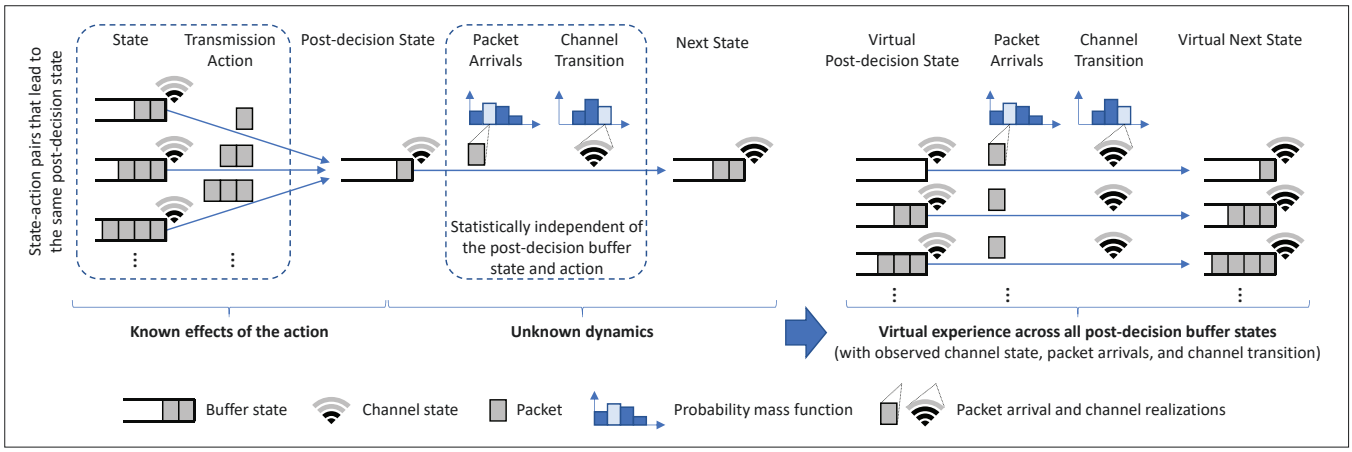


FIGURE 3. Post-decision state and virtual experience concepts applied to the example system model in Fig. 2.

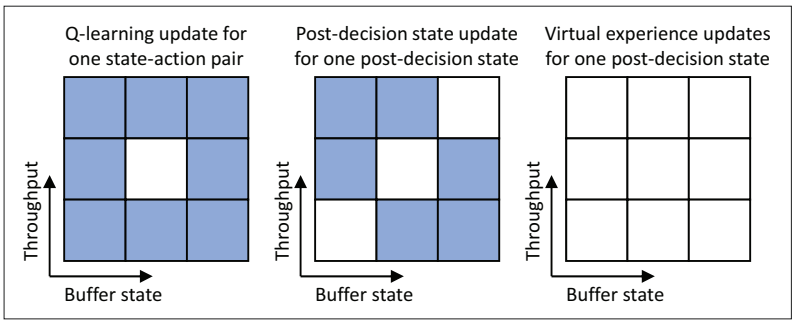


FIGURE 4. State-action pairs updated in each step (white boxes) for the example system in Fig. 2: left: Q-learning; middle: post-decision state learning; right: VE learning.

transition probabilities, which are independent of the transmission action (Fig. 3). This dramatically improves the system's performance during the learning process because we can always exploit the best known action and avoid penalties for exploring sub-optimal actions.

Second, we further establish that the PDS value function can be updated over multiple PDSs simultaneously if the unknown dynamics are statistically independent of one or more PDS variables. We refer to this as virtual experience (VE) learning because it enables updating the PDS value function over unvisited states. On the right side of Fig 3, we illustrate the VE concept in the context of the example system in Fig. 2. Here, we can apply VE updates across every post-decision buffer state in each step because the packet arrival distribution and the channel transition probabilities are independent of the post-decision buffer state. Note that VE updates are performed using the channel state, number of packet arrivals, and next channel state that were actually observed in the step.

Many existing RL algorithms update multiple states in each step (e.g., the model-free temporal-difference-lambda algorithm and the model-based Dyna-Q algorithm [8]). However, these algorithms can only update *previously visited* states because they do not leverage any system knowledge. Consequently, they operate from a "cold start" in each state, yielding sub-optimal performance. For instance, in the example system in Fig. 2, these algorithms struggle to learn how to avoid buffer overflows. In contrast, VE learning can quickly learn how to avoid buffer overflows without ever experiencing them.

## Comparison among Learning Algorithms

Figure 4 illustrates the qualitative differences between Q-learning and our advances for the example system model in Fig. 2. Here, the post-decision buffer state is calculated as the current buffer state minus the number of transmitted packets (*throughput*). We omit the channel state for simplicity. A Q-learning update on a buffer-throughput pair only modifies the action-value function for that buffer-throughput pair. A PDS learning update on a post-decision buffer state provides information about all buffer-throughput pairs that can potentially lead to it. Finally, VE learning updates provide information about every post-decision buffer state.

Figure 5 examines the quantitative performance of Q-learning and our advances over a simulated wireless link. This model extends the example system model in Fig. 2 to include a binary dynamic power management (DPM) state and a binary DPM action, such that the transmitter can be turned on and off to save energy [13]. The system is simulated over 75,000 steps with 10 ms duration. The transmitter consumes 320 mW of power in the "on" state (in addition to any transmission power), 0 mW in the "off" state, and 320 mW when transitioning between them. We assume that the buffer can hold a maximum of 25 packets of size 625 bytes and that data arrives into the buffer according to a Poisson process at a rate of 200 packets/s. The channel state (gain) evolves as a finite-state Markov chain with 8 states, ranging from −18.82 dB to −2.08 dB. Note that the finite-state Markov chain and the data arrival distribution are not known by the RL algorithms. The transmitter can transmit 0 to 10 packets in each step, with a transmission power that is convex and increasing in the number of transmitted packets, and is higher in worse channel states. We set the average buffer delay constraint to 4 packets.

After 75,000 steps, Q-learning experiences excessive overflows and is not able to stabilize the buffer. It performs poorly because it requires action exploration and only learns about one state-action pair in each step. Consequently, it struggles to learn that it must keep the transmitter in the "on" state to transmit enough packets to meet the delay constraint. PDS learning performs significantly better than Q-learning. Since

the unknown packet arrival distribution and channel transition probabilities are independent of the actions, PDS learning does not require action exploration. Moreover, as illustrated in Figs. 3 and 4, learning about one PDS provides information about all buffer-throughput pairs that can precede it. Consequently, PDS learning meets the 4-packet delay constraint within 30,000 steps, and after 75,000 steps its average power consumption is approximately 27 percent lower than Q-learning's. Finally, we are able to perform VE updates across every post-decision buffer state in each step because the packet arrival distribution and the channel transition probabilities are independent of the post-decision buffer state. This allows VE learning to meet the 4-packet delay constraint within 300 steps and to achieve near-optimal power within 3000 steps.

The benefits of PDS learning and VE come at the expense of increased *action selection* and *learning update* complexity. In Q-learning, the action selection and update steps require optimizing the action-value function over the actions, so they have complexity proportional to the number of actions. In PDS learning, both steps also require calculating an expectation over the PDS value function w.r.t. the known transition probabilities, so they have worst case complexity proportional to the product of the number of states and actions. VE has the same action selection complexity as PDS learning, but its update complexity increases linearly with the number of VE updates.

## Structure-Aware Value Function Approximation

We aim to reap the benefits of VE updates while reducing their complexity by exploiting prospective *structural properties* of the optimal value function. Such properties include, for example, integer convexity/concavity and super/submodularity [14], and can be exploited via *value function approximation*.

In [14], we used structure-aware value function approximation to optimize a delay-sensitive energy harvesting wireless sensor. The energy harvesting model extends the example model from Fig. 2 to include a battery state and energy arrivals with unknown statistics. Here, the post-decision buffer state is defined as in the original model, while the post-decision battery state is defined as the current battery state minus the energy required for transmission. We are able to perform VE updates across every post-decision buffer-battery state pair because the data and energy packet arrival distributions are independent of these PDSs.

For illustration, assuming that the buffer can hold 32 data packets and the battery can hold 32 energy packets, learning the optimal PDS value function using VE would require 1089 = (32 + 1) × (32 + 1) buffer-battery state pair updates in each step, which is computationally prohibitive. However, based on the properties of the system's cost and transition probability functions, we have established that the system's optimal PDS value function is:

- Non-decreasing and integer convex (i.e., has increasing differences) in the buffer state
- Is non-increasing and integer convex in the battery state

Leveraging this structure, we can learn an approximate PDS value function with only 18 buffer-battery state pair updates per time step (with the rest obtained via interpolation), with a maximum error of 20 from the optimal PDS value function, and at an order of magnitude lower computational/memory complexity.

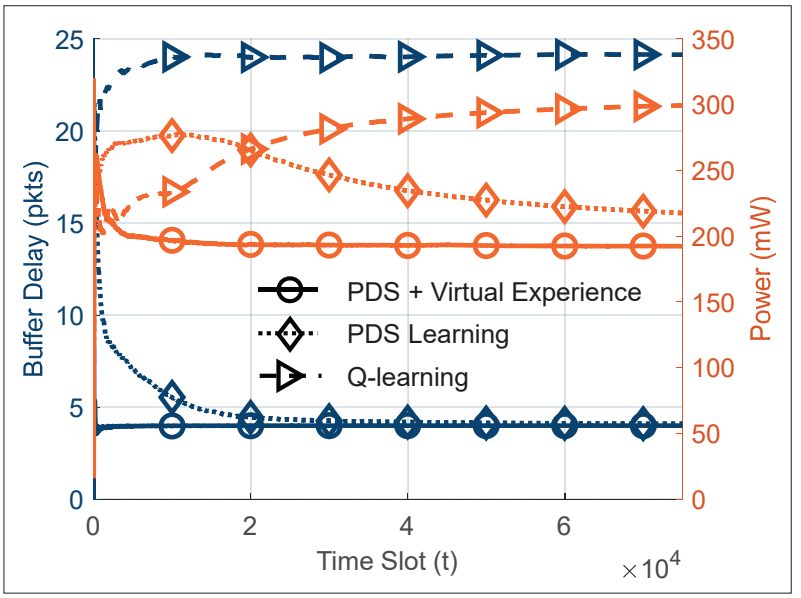


FIGURE 5. Cumulative average performance vs. time for a wireless link with dynamic power management: left axis: buffer delay (in queued packets); right axis: consumed power (in mW).

Our structure-aware value function approximation enables us to precisely control the trade-off between the induced computing/memory complexity and the associated learning approximation error by varying the approximation granularity. Due to the structural properties of the system's optimal PDS value function, the learning algorithm achieves a bounded approximation error w.r.t. the optimal PDS value function. This is in contrast to existing value function approximation-based RL algorithms, which are only guaranteed to converge under strong conditions when using linear approximations, and provide no guarantee on the approximation error.

## Extension to Multi-Agent RL

Our advances can be extended to decentralized multi-agent settings, such as those encountered in multi-access scheduling and multihop routing problems. One approach could be through a controller node that will coordinate the agents in their joint learning effort. However, the complexity of this approach could be penalizing as the number of agents increases due to the *curse of dimensionality*. An alternative could be to decompose the joint multi-agent learning problem into a collection of coupled single-agent problems that can be solved in a decentralized manner, where each agent will learn the aggregate impact of all other users on its operation. Finally, solving a joint decentralized multi-agent problem requires message passing among the agents to disseminate knowledge [15]. Sub-optimality w.r.t. centralized solutions, convergence properties, and stability of such methods will depend on the frequency of knowledge exchange among the agents and its level of accuracy and span (local vs. global, instantaneous vs. delayed, approximate vs. exact, etc.).

| Ref. | Problem | Objective | State(s) | Action(s) | Algorithm | PDS/VE possible? |
|---|---|---|---|---|---|---|
| [1] | Joint compression and transmission control for energy harvesting IoT | Minimize sum of compression and transmission energy | 1) Channel state; 2) volume of sensed data; 3) data buffer state;4) volume of harvested energy; 5) battery state | 1) Compression level; 2) number of transmitted packets | Q-learning | Yes, for both buffer and battery states |
| [2] | Uplink transmission rate control for mobile IoT | Maximize channel utilization | History of channel states | Transmission rate | DQN | No, the problem is unstructured |
| [3] | Duty cycle optimization for energy harvesting IoT | Maximize duty cycle while minimizing battery outages and duty cycle variance | 1) Battery state; 2) volume of harvested energy; 3) weather forecast; 4) previous duty cycle | Duty cycle | PPO | Yes, for the battery state |
| [4] | Multi-access control for energy harvesting IoT | Maximize sum throughput | 1) UE channel states; 2) UE battery states | UE channel assignments | DQN | Yes, for the battery state |
| [5] | Multihop ad hoc network formation for IoT connectivity | Optimize sum of network throughput and each node's transmission power | No. nodes located in transmission range | Transmission range | Double DQN | No, the problem is unstructured |
| [6] | Fresh data collection in UAV-assisted IoT | Minimize sum of the AoI across IoT sensors | 1) UAV location; 2) AoI for each IoT sensor; 3) slack in UAV lifetime; 4) slack in UAV energy | 1) UAV movement; 2) scheduled IoT sensor | DQN | Yes, PDSs can be defined for all states |
| [7] | Computation offloading for IoT edge computing | Minimize sum of power and delay | 1) Channel condition; 2) Task buffer state; 3) Remaining end-device CPU resources | 1) Offloading decision; 2) transmission power level | Q-learning | Yes, for the task buffer state |

**TABLE 1.** Related work on optimizing wireless IoT systems with RL.

## Extension to Deep RL

Our advances can also be extended to deep RL. Introducing PDSs into deep RL will allow us to use deep neural networks with fewer parameters to approximate the value function and/or policy. This will, in turn, reduce the computing resources and convergence time required to train the network(s). Interestingly, when introducing VE into deep RL, VE tuples can be added to the so-called replay buffer and used to perform the usual batch learning updates. Consequently, VE adds negligible computational costs to the learning update step, in contrast to its application in conventional RL. We expect that VE will reduce the overall time required to train deep RL models and that it will be especially useful in settings where obtaining experience is expensive.

## Related Work

RL is increasingly being used to optimize wireless IoT systems [1–7]. This state-of-the-art work takes important steps toward realizing adaptive, intelligent, and autonomous IoT systems; however, it relies on purely data-driven RL algorithms. In [2, 5], this may be the best that can be done due to the unstructured nature of the considered problems. But, in [1, 3, 4, 6, 7], we see opportunities to enhance the proposed RL algorithms using our advances. In Table 1, we summarize the aforementioned papers in terms of the considered problem, optimization objective, system states, actions, RL algorithms, and opportunities to leverage PDSs or VE.

Several of the problems include buffers to store sensed data [1], computational tasks [7], or harvested energy [1, 3, 4]. Such buffers allow for the introduction of post-decision buffer states and VE updates across all buffer states, similar to the data buffer in our example system model (Fig. 2) and the battery state in our energy harvesting model.

One of the considered problems includes actions that induce deterministic state transitions [6]. Specifically, [6] considers the problem of fresh data collection in unmanned autonomous vehicle (UAV)-assisted IoT. In their formulation, the next state variables such as UAV location and Age of Information (AoI) are deterministic functions of the action variables (i.e., the UAV's movement and the scheduling action). Such deterministic systems allow for the introduction of PDSs that are equal to the next states.

While two of these papers use Q-learning [1, 7], the others leverage deep RL algorithms, namely, deep Q networks (DQNs) [2, 4, 6], double DQNs [5], and proximal policy optimization (PPO) [3]. We see opportunity to enhance these deep RL algorithms to exploit knowledge about the underlying IoT systems. PDSs and VE offer a promising approach toward achieving this goal.

## Conclusion

RL has the potential to enable a novel class of self-organizing wireless systems that can quickly learn their optimal decision policies, as they take actions and acquire knowledge about their operating environment. We illustrate how integrating post-decision states, virtual experience, knowledge about the value function's structure, and value function approximation enables new RL algorithms that achieve fast convergence, optimality guarantees, and low compute and memory complexity. We explore the application of these techniques in the single-agent RL scenario, where there is a single decision maker, and highlight their prospective application in the context of multi-agent and deep RL scenarios. We identify opportunities for our advances to improve deep RL algorithms for wireless IoT.

## References


[1] V. Hakami *et al.*, "An Optimal Policy for Joint Compression and Transmission Control in Delay-Constrained Energy Harvesting IoT Devices," *Computer Commun.*, vol. 160, 2020, pp. 554–66.

[2] W. Xu *et al.*, "Autonomous Rate Control for Mobile Internet of Things: A Deep Reinforcement Learning Approach," *IEEE Vehic. Tech. Conf.*, 2020, pp. 1–6.

[3] A. Murad *et al.*, "Autonomous Management of Energy-Harvesting IoT Nodes Using Deep Reinforcement Learning," *Int'l. Conf. Self-Adaptive and Self-Organizing Systems*, 2019, pp. 43–51.

[4] M. Chu *et al.*, "Reinforcement Learning-Based Multiaccess Control and Battery Prediction with Energy Harvesting in IoT Systems," *IEEE IoT J.*, vol. 6, no. 2, 2018, pp. 2009–20.

[5] M. Kwon, J. Lee, and H. Park, "Intelligent IoT Connectivity: Deep Reinforcement Learning Approach," *IEEE Sensors J.*, vol. 20, no. 5, 2019, pp. 2782–91.

[6] M. Yi *et al.*, "Deep Reinforcement Learning for Fresh Data Collection in UAV-Assisted IoT Networks," *IEEE INFOCOM Wksp.*, 2020, pp. 716–21.

[7] X. Liu, Z. Qin, and Y. Gao, "Resource Allocation for Edge Computing in IoT Networks via Reinforcement Learning," *IEEE ICC*, 2019, pp. 1–6.

[8] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[9] A. Roy, H. Xu, and S. Pokutta, "Reinforcement Learning Under Model Mismatch," arXiv preprint arXiv:1706.04711, 2017.

[10] E. Altman, *Constrained Markov Decision Processes*, vol. 7, CRC Press, 1999.

[11] N. Salodkar *et al.*, "An On-Line Learning Algorithm for Energy Efficient Delay Constrained Scheduling Over a Fading Channel," *IEEE JSAC*, vol. 26, no. 4, 2008.

[12] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *J. Artificial Intelligence Research*, vol. 4, 1996, pp. 237–85.

[13] N. Mastronarde and M. van der Schaar, "Fast Reinforcement Learning for Energy-Efficient Wireless Communication," *IEEE Trans. Signal Process.*, vol. 59, no. 12, 2011, pp. 6262–66.

[14] N. Sharma, N. Mastronarde, and J. Chakareski, "Accelerated Structure-Aware Reinforcement Learning for Delay-Sensitive Energy Harvesting Wireless Sensors," *IEEE Trans. Signal Process.*, vol. 68, no. 1, Feb. 2020, pp. 1409–24.

[15] N. Coutinho *et al.*, "Dynamic Dual-Reinforcement-Learning Routing Strategies for Quality of Experience-Aware Wireless Mesh Networking," *Computer Networks*, vol. 88, Sept. 2015, pp. 269–85.


## Biographies


Nicholas Mastronarde [SM] received his Ph.D. degree in electrical engineering from the University of California at Los Angeles. He is an associate professor in the Department of Electrical Engineering, University at Buffalo.

Nikhilesh Sharma received M.S. and Ph.D. degrees in electrical engineering in 2017 and 2020, respectively, from the University at Buffalo.

Jacob Chakareski [SM] completed his Ph.D. degree in electrical and computer engineering at Rice University and Stanford University. He is an associate professor in the Ying Wu College of Computing, New Jersey Institute of Technology, where he holds the Panasonic Chair of Sustainability.