# ACES - Automatic Configuration of Energy Harvesting Sensors with Reinforcement Learning

FRANCESCO FRATERNALI, University of California, San Diego, USA
BHARATHAN BALAJI, Amazon, USA
YUVRAJ AGARWAL, Carnegie Mellon University, USA
RAJESH K. GUPTA, University of California, San Diego, USA

Many modern smart building applications are supported by wireless sensors to sense physical parameters, given the flexibility they offer and the reduced cost of deployment. However, most wireless sensors are powered by batteries today and large deployments are inhibited by the requirement of periodic battery replacement. Energy harvesting sensors provide an attractive alternative, but they need to provide adequate quality of service to applications given the uncertainty of energy availability. We propose ACES, that uses reinforcement learning to maximize sensing quality of energy harvesting sensors for periodic and event-driven indoor sensing with available energy. Our custom-built sensor platform uses a supercapacitor to store energy and Bluetooth Low Energy to relay sensors data. Using simulations and real deployments, we use the data collected to continually adapt the sensing of each node to changing environmental patterns and transfer learning to reduce the training time in real deployments. In our 60 node deployment lasting two weeks, nodes stop operations for only 0.1% of the time, and collection of data is comparable with current battery-powered nodes. We show that ACES reduces the node duty-cycle period by an average of 33% compared to three prior reinforcement learning techniques, while continuously learning environmental changes over-time.

CCS Concepts: • **Computer systems organization** → **Sensor networks**; • **Computing methodologies** → **Reinforcement learning**; • **Hardware** → *Renewable energy*; *Sensor applications and deployments.*

Additional Key Words and Phrases: Internet of Things, Automatic Configuration, Reinforcement Learning, Smart-Buildings, Energy Harvesting, Battery-Less, Real Deployment

**ACM Reference Format:**
Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, and Rajesh K. Gupta. 2019. ACES - Automatic Configuration of Energy Harvesting Sensors with Reinforcement Learning. *ACM Trans. Sensor Netw.* 1, 1, Article 1 (January 2019), 31 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Buildings are an essential part of modern society and benefit immensely from the Internet of Things (IoT) technologies [24]. Networked sensors are the bedrock of modern building services such as security, fire safety, energy, and lighting. Pervasive sensors deployed across a modern building can sense temperature, light, smoke, occupancy, energy

Authors' addresses: Francesco Fraternali, frfrater@eng.ucsd.edu, University of California, San Diego, San Diego, USA; Bharathan Balaji, Amazon, Seattle, USA, bhabalaj@amazon.com; Yuvraj Agarwal, Carnegie Mellon University, Pittsburgh, USA, yuvraj@cs.cmu.edu; Rajesh K. Gupta, University of California, San Diego, San Diego, USA, gupta@eng.ucsd.edu.

use; it is possible to have hundreds to thousands of sensors in a modern office building [17]. While traditional buildings use wired sensors, wireless technology is being increasingly adopted due to lower deployment cost and flexibility of placement [19]. Much of the wireless sensors in the market are battery powered [43] and manual battery replacement is a key limitation that inhibits large scale deployments [27]. Energy harvesting sensors provide an attractive alternative, but their design needs to ensure adequate Quality of Service (QoS) given the limited and often uncertain energy availability. Many innovative battery-free solutions have therefore been proposed in prior works [8, 9, 21, 47, 55].

Energy harvesting systems have to make a careful trade-off across sensing, communication, and computation requirements to maximize application utility with available energy [15, 28]. These design tradeoffs change depending on hardware, application requirements and energy availability in the environment. Prior works either perform manual configuration or use heuristics to identify the operating points [2, 21, 49, 59, 65]. For example, EnOcean launched a commercial energy harvesting sensor in June 2019 that uses *"a simple user interface consisting of one button and one LED allows for simple configuration without additional tools"* [65]. In our prior work, we designed the Pible platform[21], with the goal of harvesting enough energy to last an entire day of operation. However, the key limitation of Pible is that its operational configuration was static and configured manually. It was, therefore, unable to adapt each sensor to the different environmental conditions that affect the amount of energy harvested. Manual configuration does not scale and heuristics do not generalize well to every context.

To overcome this limitation, we present our system *ACES* that uses Reinforcement Learning (RL) [54] to automatically optimize the operation of sensors nodes to maximize QoS under uncertain energy availability. In RL, an agent interacts with an environment and learns to make intelligent decisions with experience. A domain expert identifies the objective (i.e. reward function in RL terminology) and the inputs (i.e., state) that affect the decisions (i.e. actions) of the agent. The agent tries out different actions and learns from the feedback (rewards) received. RL algorithms are good at learning sequences of actions that maximize the long term expected cumulative rewards. Several reasons make RL an appropriate solution for this problem [15, 22, 30]: (i) *automation*, by learning *directly from experience*, RL can offer an alternative to heuristic-based approaches. An agent starts by knowing nothing about the environment and the task to be performed, but learns to make better decisions by interacting with the environment; (ii) *optimization*, in [40] authors show that RL is comparable or better than ad-hoc heuristics. Given these attributes, RL is the perfect candidate to address the challenges presented earlier since we want a system that can detect environmental patterns (i.e. human presence, indoor lighting) whenever present, and automatically configure sensors over longer periods of time.

Hence, in case of energy harvesting, the long term objective ensures that it learns patterns in energy availability across days, nights and weekends. Energy availability is not perfectly predictable, especially in indoor conditions. The RL agent learns a strategy and is robust to noise (e.g. due to varying energy availability). However, learning an RL policy by directly trying out different actions in the real world can be time-consuming, convergence can take several weeks and exploratory actions can lead to poor QoS during training. ACES exploits a simulator that uses data from actual deployments to reduce convergence time and show that the policies learned by the simulator work demonstrably well once deployed in the real world at scale. Using an Intel Core-i7 CPU clocked at 3 GHz, our algorithm converges in <30 minutes of wall-clock time, making it feasible to use ACES in the real world for applications in which information dynamics vary hourly/daily.

We implemented and evaluated ACES in a real deployment of energy harvesting sensors in our Computer Science Department building at UC San Diego. We built a Bluetooth Low Energy (BLE) based energy harvesting sensor platform, called Pible [21], that uses a solar panel to harvest energy and a supercapacitor to store energy. We use the node to sense light periodically and detect motion events with a PIR sensor. In smart buildings, having powered base station

relays is typical [3]. The node sends the data to the closest base station, hence in this work, we only target one-hop sensor networks. We categorize 5 types of indoor lighting conditions and deploy a sensor node for each location type. We collect measurements such as light intensity and supercapacitor voltage level. Using these data traces, we develop a simulator that models the essential aspects of our sensor-node in different environments. We use the simulator to train the ACES RL agent using the Q-learning algorithm [63]. We then perform multiple deployments for subsequent evaluations that traded off learning a one-time policy from historical data versus learning a new policy each day to better detect environmental diurnal patterns. Our real-world deployments show that ACES effectively learns from observing environmental changes and appropriately configures each sensor node to maximize sensing quality while avoiding energy storage depletion.

To the best of our knowledge, ours is the *first real-world deployment* that demonstrates an RL based sensing mechanism for energy harvesting sensors in indoor environments. We deploy 5 nodes, each exposed to different lighting conditions for 15 days, and use the light measurements collected to learn a policy in our simulator. We then deploy the learned policy in the same nodes and let the agent take action for 31 days. The nodes achieved a mean sampling period of 56 seconds, opportunistically collecting up to 1.7x more data than battery-powered sensors when energy is available and had 0% dead time across nights and weekends. However, a one-time learning of a policy requires a data collection phase and can be susceptible to changes in the environment. Hence, we introduce a new training strategy that *does not require any historical data*. We start with a default periodic policy on the first day and then train a new policy each day using data collected from all the previous days. This 'day-by-day' training strategy learns a stable policy within a few days and achieves near-zero dead time. To alleviate poor performance in the initial training phase of a few days, we further reduce training time with transfer learning by training an initial policy using data collected by sensor nodes in 5 different lighting conditions. Finally, we generalize ACES formulation to event-driven sensing. We deployed 45 nodes with PIR event sensing and periodic light sensing for two weeks. ACES nodes could detect 86% of the events on average compared to a baseline of battery-powered nodes which detected all events. We expanded our deployment to 15 more nodes with periodic light sensing to validate the findings of our initial 5 node deployment.

We compare ACES with three RL Q-learning based methods proposed in prior works and a local-mote heuristic algorithm. In a week-long simulation experiment, we show that ACES reduces the node duty-cycle period by 33% on average w.r.t. prior RL techniques, while continuously learning environmental changes over-time. ACES is open source, and all the software is available online [1].

## 2 RELATED WORK

A multitude of innovative energy harvesting solutions has been proposed in the literature [14, 46, 53, 58]. In several works, the sensors use up energy as it becomes available [9, 10, 14, 46], e.g., harvesting AC power lines for energy metering [14], RFID based battery-free camera [46], thermoelectric harvesting based flow sensor [41]. Backscatter sensors are a special case that eliminate communication-based energy expenditure by using existing RF signals for both communication and harvesting [16, 36, 55]. However, sensing whenever energy is available is not ideal for all applications - sensors may miss important events because they do not store enough energy or send too much data when it is not needed [33, 37]. A solar panel powered temperature sensor should work on nights and weekends even when energy availability is low and a backscatter-based motion sensor should capture events when there are no RF

[1] https://github.com/francescofraternali/ACES.git

transmissions. Hence, these sensors nodes need to be carefully configured to increase their quality of service and compete with existing battery-based solutions.

Other related work tackles the intermittent operation of battery-less energy harvesting systems themselves [25, 26, 38, 39]. In contrast, our work is looking to solve an orthogonal problem, where we are optimizing our system to perform sensing, computation, and communication while managing limited resources.

Reducing manual intervention for sensor configuration is an important task as underlined by many works [11, 15, 21, 29, 31, 45, 57]. Adaptive duty cycling of energy harvesting sensors has been used to achieve energy-neutral operations [28, 35, 45, 60]: nodes adjust their duty-cycle parameters based on the predicted energy availability to increase lifetime and application performance [53]. Moser et al. [45] adapt parameters of the application itself to maximize the long-term utility based on future energy availability prediction. Reinforcement learning (RL) also predicts energy availability, but unlike the above solutions, it also learns an optimal policy that maximizes the long-term reward. Hence, it makes better decisions compared to utilizing heuristics as we show in our evaluation.

Several works have the goal to maximize the overall Quality of Monitoring (QoM) in wireless rechargeable sensor networks (WRSNs) [13, 64, 66]. Authors in [13, 66] investigate the sensor scheduling problem aiming to maximize QoM for events capture. They solve a much harder problem: detecting events with a collection of sensors. While their work provides theoretical guarantees, it either assumes events are completely random or have specific distributions. Results are based on simulations and authors did not consider noise variability in real-world data. We focus on a pragmatic solution for a collection of independent sensing nodes for periodic sensing with harvesting. Our work is experimental, where we use real data and real sensors in a building. Our RL algorithm is agnostic to the event distribution as the agent learns the distribution of events based on past data. We have solved a number of issues in the deployment of RL in practice with day-by-day learning and transfer learning.

Zhenchun et al. [64] use RL to autonomously plan and adjust the charging path of mobile chargers, to charge more efficiently in a sensor network. Our goal is orthogonal to this problem, as we only rely on solar-based charging and do not need mobile chargers. We show that even without using batteries we can achieve almost 0% dead time, making it a much simpler and cheaper solution than [64].

RL has been identified by several prior efforts as a mechanism to configure wireless sensors and has shown promising results based on simulations [4, 15, 30, 67]. Zhu et al. [30] apply RL for sustaining perpetual operations and satisfying the throughput demand requirements for energy harvesting nodes (i.e. RLTDPM). However, their algorithm is built and tested in an outdoor environment, where sunlight patterns are consistent throughout the day (light is available from sunrise to sunset). We focus on indoor sensing where the daylight patterns are less well-defined and the light availability is affected by human occupancy. With respect to our work, RLTDPM is a one-time learning, and even if it can adjust its sensing to application's requirements, it is not able to learn changes in the environment over-time. Shaswot et al. [52] use solar energy harvesting sensor nodes powered by a battery to teach an RL system to get energy-neutral operations. They use the SARSA algorithm [54] to study the impact of weather, battery degradation and changes to the hardware. However, their work is also limited to simulations of outdoor environments.

Simulation results by Dias et al. [15] optimize energy efficiency based on data collected over five days from five sensor nodes using on-line reinforcement learning with Q-Learning (i.e. On-Line RL). However, their reward function does not depend on battery level or energy consumed, and thus, does not capture realistic conditions. We compared ACES with On-Line RL, showing that our system can reduce nodes's duty-cycle period by up to 35%. They also assume a fixed 12 hours period as the time needed by the Q-Learning algorithm to learn the action-value function for the rest of the 4.5 days experiment. However, a one-time training scheme is unable to capture all kinds of environmental

changes. In our work, we propose a periodic training method that maximizes system performance even in the presence of environmental changes.

Aoudia et al. present RLMan [4] that uses the actor-critic algorithm with linear function approximation [23]. They use existing indoor and outdoor light measurements for simulations. While the authors claim their linear function approximations facilitate RL training in a wireless node, they do not deploy RL on a real node, nor do they report their memory and compute requirements. Our formulation performs RL training on a local server and eliminates the need for putting RL inside the node. We use the Q-learning algorithm, and our Q-table is only 25kB in size and demonstrate that it can be embedded in the sensor node as well. Our prior work [22] used the Q-learning algorithm for adapting the sampling rate. The paper focused on methods to scale RL to large deployments and reduce training time. Using simulations, we showed that training a new policy periodically based on real light data improved the adaptability to changing conditions. We also showed that transfer reinforcement learning works well and it is possible to share the learned policies among sensor nodes that share similar lighting conditions. However, all of our results were based on simulations from 5 sensor nodes in multiple lighting conditions and were not deployed or tested in the real world. In this work, we show that these results transfer to the real-world with multiple large scale deployments. We have also extended our ACES RL formulation to event-driven sensors and evaluate their performance in real deployments. Besides, we analyze the effect of changing the state and action space and quantify the energy neutrality of the learned policies.

To the best of our knowledge, our work is the first to exploit reinforcement learning for adaptive sampling in energy harvesting nodes in a *real deployment*. The RL design needs to take into account changing environmental conditions, imprecise state estimates, and stochastic state transitions. We present a formulation of our problem and perform simulation-based modeling to capture realistic conditions that transfer to the real world. Furthermore, we have extended our problem formulation to accommodate *event-driven sensors* such as PIR motion detectors. The sensor node needs to preserve enough energy to communicate events as they occur. We show that ACES successfully learns typical event firing patterns and curtails its sensing period to account for the additional energy use.

Table 1. Features Comparison of ACES w.r.t. Prior Techniques

| | Learn Over-Time | Deployment Time | Algorithm Execution in the Node | Tested in Real-World |
|---|---|---|---|---|
| Mote-Local [21] | No | ~ 0 | Yes | Yes |
| RLMan [4] | No | Days | Yes | No |
| On-Line RL [15] | No | Hours | No | No |
| RLTDPM [30] | No | Days | No | No |
| WRSN [13] | No | ~ 0 | No | No |
| ACES | Yes | ~ 0 with TL | Yes | Yes |

Legend: TL = Transfer Learning

Table 1 reports the main differences between current state-of-the-art techniques including three RL methods and ACES. As we can see, ACES is the only work that is tested in the real-world while using RL. It overcomes typical real-world problems such as continuously adapting to new changes in the environment and reducing deployment-time to almost zero due to transfer learning. Furthermore, ACES can limit system failures due to network disconnections by executing the learned algorithm inside the node (Section 6).

## 3 ACES DESIGN AND IMPLEMENTATION

### 3.1 Problem Statement

In buildings, sensors are used for applications such as environment control (e.g. air conditioning), safety, security or convenience. We broadly categorize the sensing as periodic (e.g light intensity sensor) or event-driven (e.g. motion sensor). For periodic sensors, the higher the frequency, the more responsive the control systems and better the QoS. For event-driven sensors, reducing the number of missed events translate to higher QoS. However, the QoS is extremely poor if the sensors are not operational for hours at a time - control systems will lose their feedback loop, event-driven applications will be non-functional. Hence, we define our objective as to minimize the average duty cycle period while ensuring the energy harvesting nodes remain alive. As an example, a sensor with an average duty cycle period of 10 seconds w.r.t one with a 50 seconds duty cycle period, will send 5 times more data.

Nodes can be placed in different locations in a building. Each sensor node will be subject to different lighting patterns that are determined by human behavior (e.g. lights turned on and off) or by natural light whenever the node is close to a natural source of light (e.g. window). Light availability will vary from weekdays to weekends, from winter to summer and with changes in usage patterns, e.g. a conference room vs a lobby. The sensor node needs to adapt itself to these changing conditions to maximize the utility of its applications. A node placed near a window and subject to a source of natural sunlight can still be considered as an "indoor" case for two main factors: (i) indoor light can still impact the energy gathered, (ii) blinds are common and reduce outside lighting.

### 3.2 Reinforcement Learning and Q-Learning

In a typical RL problem [54], an agent starts in a state $s$ and by choosing an action $a$, it receives a reward $r$ and moves to a new state $s'$. This process is repeated until a final state is reached. RL agent's goal is to find the best sequence of actions that maximizes the long term reward. The way the agent chooses actions in each state is called its policy $\pi$.

$$s \xrightarrow[\pi]{a} r, s' \tag{1}$$

For each given state $s$ and action $a$, we define a function $Q(s, a)$ that returns an estimate of a total discounted reward we would achieve by starting at state $s$, taking the action $a$ and then following a given policy $\pi$ till a final state is reached.

$$Q_\pi(s, a) = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots \tag{2}$$

where $\gamma \leqslant 1$ is called a discount factor and it determines how much the function $Q$ in state $s$ depends on the past actions (i.e. how long in the past does the agent see) since each member in the equation exponentially diminishes the further they are in the past. Equation (2) can be rewritten in a recursive form called the Bellman equation:

$$Q_\pi(s, a) = r_0 + \gamma(r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots) = r_0 + \gamma max_a Q_\pi(s', a) \tag{3}$$

The Q-learning algorithm starts with a randomly initialized Q-value for each state-action pair and an initial state $s_0$. An episode is defined as a sequence of state transitions from the initial state to the terminal state. The algorithm follows a $\epsilon$-greedy policy, where for each state it picks the action that has the maximum Q-value with probability $(1 - \epsilon)$ and a random action otherwise. The reward obtained by selecting the action is used to update the Q-value with a small learning rate. Under the conditions that each of the state-action pairs are visited infinitely often, the Q-learning algorithm is proven to converge to the optimal function $Q^*$ [62]. The optimal policy $\pi^*$ takes the action that maximizes $Q^*$ in each state. The $\epsilon$-greedy policy is used as an exploitation exploration trade-off [54], where the occasional random

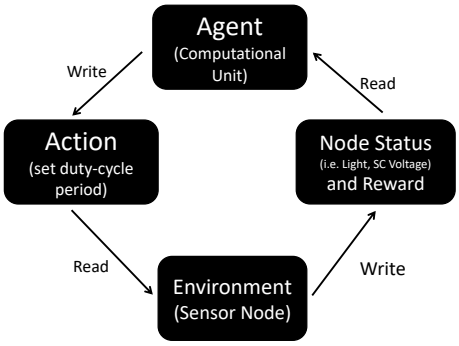


Fig. 1. Reinforcement Learning Communication Process for Energy Harvesting Sensors: Block Diagram.

actions encourage the agent to explore the state-action space. It is typical to use a high value of $\epsilon$ at the start of training and gradually reduce it over time to increase exploitation.

Between different RL algorithms, we choose Q-learning because it is easy to use when there is a set of discrete states and actions. Q-learning is an off-policy algorithm, which means we can learn a policy (and a Q function) with historical data when available. By exploiting the off-policy nature of Q-learning, we introduce two additional variants to the algorithm to reduce convergence time: day-by-day learning and transfer learning. In day-by-day learning, we learn a new Q function each day based on data collected in the past day. The Q function gradually improves each day and can accommodate changing environmental conditions. With transfer learning, we use a Q function learned for another sensor node as an initial policy for a new node.

Q-learning is guaranteed to converge to an optimal solution under certain conditions: (1) we accurately know the dynamics of the environment; (2) we have enough computational resources to complete the computation of the solution; and (3) the Markov property. Hence, in our case, Q-learning is not guaranteed to converge to an optimal solution since as we show in Section 3.3 there are a number of problems such as: (1) we discretize the state/actions and make assumptions on the energy consumption of the components; (2) we have limited computational resources and training time. Therefore, there could be some states that are not seen during training. Even with these approximations, we show that our system can find a solution that outperforms state-of-the-art results.

### 3.3 RL Problem Formulation for Energy Harvesting Sensors

Figure 1 shows an overview of our RL formulation for the configuration of energy harvesting sensor nodes. A sensor node collects energy from a solar panel and sends sensor data to a computational unit that runs the RL policy and determines the node sleep time (i.e. the time the node stays in sleep mode). The computational unit can be the node, a base station, a local server or in the cloud as long as they have enough computational power to train RL policies. We run the training on the local server.

**Objective and Quality of Service (QoS):** the objective of our problem is twofold:

- Maximize sensing (reduce duty-cycle period by reducing the node sleep time) so we capture periodic sensor measurements more frequently and minimize missed events for event-driven sensors.
- Minimize dead time. If the sensor node dies, it enters a cold start phase and could not send data for hours.

Hence, we define the Quality of Service (QoS) as the duty cycle period averaged across a period of time (e.g. 7 days):

$$\text{Duty-cycle-period} = \frac{1}{\tau}\sum_{t=0}^{\tau} \mathrm{T}_{sensing} + \mathrm{T}_{sleep} \tag{4}$$

where, $\mathrm{T}_{sensing}$ is a constant and includes the total time the node remains active (i.e. the time the node wakes-up, read sensor data, send it over BLE and goes back to sleep), $\mathrm{T}_{sleep}$ is the node sleep time (i.e. MCU in sleep mode) and $\tau$ is the total number of duty cycles in a given period. For event-driven sensors (e.g. motion sensor), a sensor is left awake during $\mathrm{T}_{sleep}$ until an interrupt is raised due to an event. The RL agent decides the $\mathrm{T}_{sleep}$ period for both types of sensors. Another metric commonly used in literature is duty cycle ratio [5]. We can compute the duty cycle ratio using the proposed duty cycle period with the following equation:

$$\text{Duty-cycle-ratio} = \frac{T_{sensing}}{\text{Duty-cycle-period}} \tag{5}$$

**Agent:** The agent is the program in the computational unit that takes in measurements from the sensor, outputs the node sleep time (i.e. $\mathrm{T}_{sleep}$) to use and updates the RL policy based on rewards.

**Environment:** It is everything outside the agent, which includes sensor nodes, the wireless channel, the lighting conditions and events that trigger the sensors.

Table 2. Node Sleep Time based on Action Index. One of the objective of our system is to reduce the *Duty-Cycle Period* of the system by reducing the Node Sleep Time. With *Duty-Cycle Period* we indicate the time the node stays in sleep mode (i.e. $\mathrm{T}_{sleep}$) plus the time it remains active (i.e. $\mathrm{T}_{sensing}$) such as waking-up, reading the sensors and transmitting the data using a wireless protocol.

| Action Index | Node Sleep Time [s] |
|---|---|
| 3 | 15 |
| 2 | 60 (1 min) |
| 1 | 300 (5 min) |
| 0 | 900 (15 min) |

**State:** We use: *(i)* light intensity, *(ii)* energy storage level, *(iii)* weekend/weekdays. We discretize light intensity and energy storage levels to 10 values each. The discretization helps reduce the state space, and hence, decreases the convergence time of the Q-learning algorithm.

**Action:** To change the duty cycle period, from 4, we can only act on $\mathrm{T}_{sleep}$ since $\mathrm{T}_{sensing}$ is dependent on the sensing characteristics and the communication mechanism. Therefore, we discretize the node sleep time to meet typical commercial duty-cycle periods for IoT devices in buildings, since they are in the order of minutes but can range from tens of seconds to an hour [1, 18, 32]. We report our node sleep time discretization in Table 2. For periodic sensors, the action selects the node sleep time to use, e.g. action 2 corresponds to a sleep time of 1 minute. For event-driven sensors, we observe that once an event is triggered, a subsequent event within a short amount of time is inconsequential. Hence, we keep the node alive until an event occurs, after which it sends the event packet and sleeps for the period indicated by the action. Any event during sleep time is missed.

**State Transitions:** The agent observes state transitions and takes actions, i.e. sends commands to change the node sleep time, every 15 minutes. A small timestep for state transitions increases the communication overhead between the base station and sensor node and a large timestep misses the opportunity to tune the sensing rate in a fine-grained manner. We select a timestep of 15 mins as a design trade-off between these factors. We use 24 hours as our episode, starting when the node is turned on. We will consider dynamic transitions based on node sleep time as future work.

**Reward Function:** The reward function is a trade-off between maximizing sensing that consumes energy while penalizing dead time. We assign rewards as follows:

- Reward = Action index (i.e. 0, 1, 2 or 3)
- Reward = -300 if energy storage level reaches 0.

If the energy storage reaches 0, the system should receive a negative reward that dissolves all the benefit of using the maximum index action available (i.e. action 3) since it could cause a dead time lasting several hours. Therefore, with the above reward function, the negative reward should be compensating for the maximum total reward in 24 hours. The following formula makes this calculation:

$$Reward <= -(\text{number of state-transitions per day} * \text{max action} = 24 * 4 * 3 = 288) \quad (6)$$

Using Formula (6), we picked -300 as a negative reward. The general principle here is to identify the objective function of the problem and formulate it as a reward function. Any constraints, such as avoiding the depletion of energy in storage, can be expressed as a negative reward. While we use a heuristic (i.e. Formula 6) to identify the negative reward coefficient, one can also identify the coefficient automatically using Lagrangian relaxation [7]. Q-learning requires the use of discrete states and actions, and the range of actions can be decided based on the desired application requirements. Fine-grained discretization can lead to a better policy, but increases convergence time. We study design trade-off between different discretizations in Section 4.2.2. Our problem formulation can be easily adapted to different sensors, energy harvesters, and applications.

**Algorithm 1:** RL Algorithm for Energy Harvesting Sensors

1: Initialize $q_{table}$ as an empty set
2: Initialize control action a, state $s_{curr}$, $s_{next}$ *time passed* = 0
3: $\epsilon = \epsilon_{min}$
4: $s_{curr} \leftarrow$ Sense Environment
5: **while** *time passed* < *episode_duration* **do**
6:

$$a = \begin{cases} \text{if (uniform(0,1) <= } \epsilon\text{) take: } argmaxQ(s_{curr}, a') \\ \text{otherwise: take a random action} \end{cases}$$

7: wait for $T$ time units, *time passed* += $T$
8: $s_{next} \leftarrow$ Sense Environment
9: r = reward($s_{curr}$, a, $s_{next}$)
10: $q_{predict} = q_{table}[s_{curr}, a]$
11: $q_{target} = r + \gamma * max(q_{table}[s_{next}, \forall\ a])$
12: $q_{table}[s_{curr}, a] += \alpha * (q_{target} - q_{predict})$
13: $\epsilon_{min} = \epsilon_{min} + \Delta$
14: $\epsilon = min(\epsilon_{max}, \epsilon_{min})$
15: $s_{curr} \leftarrow s_{next}$
16: **end while**

The specific values of each variable used are given in Table 3.

### 3.4 RL Algorithm

Algorithm 1 details the use of Q-learning in ACES. After initialization (line 1-3), the agent receives the node state - light, energy storage voltage and current action index (line 4). The algorithm starts the episode and selects an action

following the $\epsilon$-greedy policy (line 6). In the beginning, $\epsilon$ is initialized to $\epsilon_{min}$, the algorithm selects a lot of random actions in the first phase of the learning since uniform() produces a random value uniformly distributed between 0 and 1. At each time step (15 mins), $\epsilon$ is increased by $\Delta$ (line 13-14), causing the action policy to select more exploitative actions over time.

The sensor node updates its node sleep time and sends its state again after $T$ time units (line 7-8). We calculate the rewards with the current state, action and next state (line 9). We assume our system has reached convergence when the mean of all the values in the Q-Table does not change their value by >5%.

We progressively improved our policy training strategy to reduce time to convergence.

**One-Time Learning:** We train a policy in a simulator based on sensor measurements collected for 15 days. This is a typical setting used in prior works [4, 15].

**Day-by-Day Learning:** We train a policy every day based on the sensor measurements collected in the past 24 hours. Each day's training starts with the policy learned in the previous day. The policy converges within a few days to achieve energy-neutral operations. The daily training also adjusts the policy to changing environment conditions (non-stationary environments). This borrows from Batch RL methods in literature [34].

**Transfer Learning:** Instead of learning a policy from scratch, the initial policy is borrowed from another node's converged policy. We achieve 0% dead time from the first day of deployment with transfer learning. We continue to use the day-by-day learning procedure for iterative improvement of the policy.

Figure 2 shows an overview of one-time, day-by-day and transfer learning methods. Day-by-day learning assures continual learning to changing conditions and removes a separate data collection period. Transfer learning reduces dead time during initial phase of the deployment.

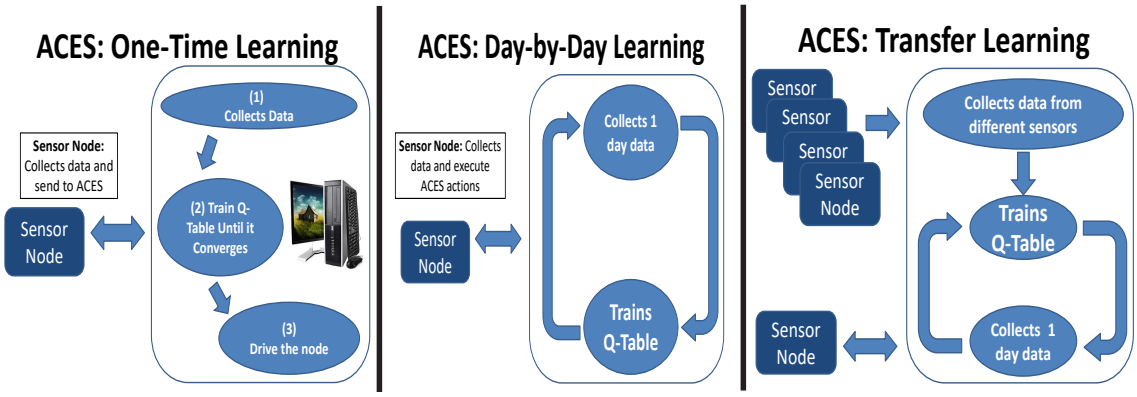


Fig. 2. A comparative overview of the one-time, day-by-day and transfer learning methods used in ACES. Day-by-day learning assures continual learning to changing conditions and removes a separate data collection period. Transfer learning minimizes dead time during initial phase of the deployment.

Table 3 reports the hyper-parameters used for the Q-learning algorithm for our simulations.

### 3.5 Hardware and Communication Process

*3.5.1 Sensor Node:* We developed a general-purpose energy harvesting battery-less sensor (Figure 3) [20, 21]. We use a solar panel (AM-1454 [12]) as our energy harvester and store energy in a super-capacitor [48]. Once the voltage reaches

Table 3. Q-Learning Hyper-parameters used for simulations

| Hyper-Parameter | Value |
|---|---|
| Reward-decay ($\gamma$) | 0.99 |
| Epsilon max ($\epsilon_{max}$) | 1 |
| Epsilon min ($\epsilon_{min}$) | 0.1 |
| Epsilon increment ($\Delta$) | 0.0004 |
| Learning rate ($\alpha$) | 0.1 |
| Episode Duration | 24 hours |
| Wait Time $T$ | 15 mins |
| Training phase | 15 days |

a usable level, an energy management board powers the microcontroller (MCU) to start its operations. We use a 1F super-capacitor with a 5.5V nominal voltage. The MCU uses a high resistance voltage divider (10MOhm) to minimize leakage current. We use BLE for communication and the node has light and PIR sensors. The sensor-node achieves up to a week of lifetime without light when it sends one sensor measurement every 10 minutes.


Fig. 3. General Energy Harvesting Sensor-Node.

*3.5.2 Wireless Sensor Network Architecture:* The base station uses BLE 4.2 gattool functions to exchange data with the sensor nodes deployed around the building. As soon as a sensor-node wakes-up, it starts advertising. The base station reads the advertisement, connects to the sensor-node and exchanges data. During the connection, the base station communicates the next action to do to the sensor node while the sensor node communicates the read sensor values (i.e. light, temperature, PIR, SC Voltage) to the base station. The base station stores and sends the data to a local server for post-processing using a Wi-Fi connection. In this work, the base station is only used for passing the data from the sensor nodes to the local server. The RL training is done at the local server and the sensor nodes execute the actions decided. Figure 4 shows all the steps we just described. Due to the nature of BLE 4.2 that we use, we can only target one-hop sensor networks. Our methods can be easily extended to low bandwidth networks such as low-power wide-area networks (LPWAN). From [42] the maximum payload length for typical LPWAN protocol ranges from 8 bytes (e.g. SigFox [42]) to up to 1600 bytes (e.g. NB-IoT [42]). Therefore, our approach can be extended to LPWAN networks since our bandwidth requirements are low as transferring data of 6 bytes. We leave empirical verification for future work. In our system, the base station is composed of a Raspberry Pi equipped with a BLE USB dongle. In our deployment, we connected up to 15 nodes to a single base station. To facilitate large deployments, the nodes do not

remain connected to the base station as in a typical Bluetooth connection but they always disconnect and reconnect using the advertisement. In this way, we are not limited by the number of simultaneous BLE connections.

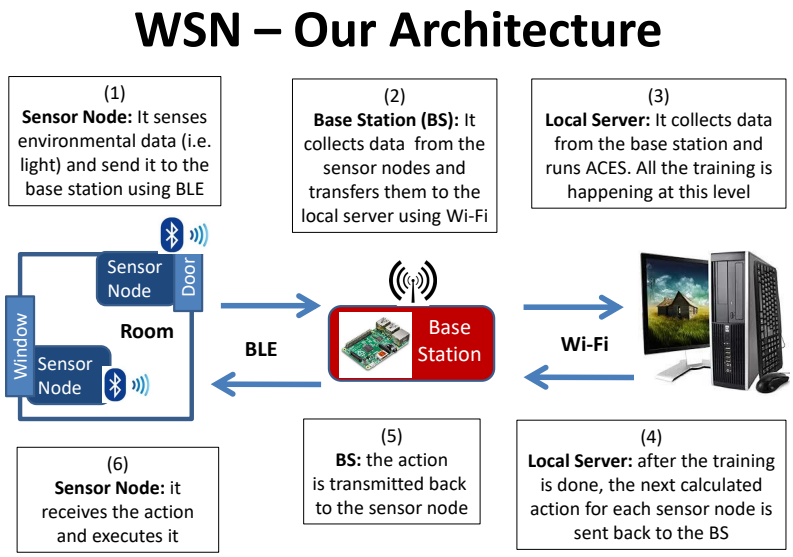


Fig. 4. Our WSN architecture. The training happens at the local server and the base station is used to transfer the data from the sensor nodes to the local server and vice-versa.

## 4 SIMULATIONS

We use the simulator to speed up the RL training since it can take thousands of episodes to converge. Our objective is to model the environment to sufficiently capture real-world characteristics while keeping its complexity low to allow for fast simulations.

### 4.1 Modeling the Simulation Environment

When the agent acts on the environment, the simulation needs to respond with the next state and reward defined in Section 3.3. To create a simulation environment, we need to identify how much energy will be consumed and harvested under a given environment condition and sensing quality. We start with modeling the sensor platform, where we need to identify the energy consumed in different modes of operations. The energy consumption can be calculated from the datasheet of individual components used or by direct measurements in different operating modes [61]. We use a combination of both. For more complex platforms where individual component analysis is not feasible, we can fit a model based on power consumption in various modes of operation [50]. The energy gained is a function of both the efficiency of the harvester module and the energy available in the environment. We use raw light measurements collected by sensors in different settings to capture environment characteristics.

*4.1.1 Current Measurements:* We use the power consumption of the solar panel and PIR from their respective datasheets and measure the current consumption for the other components to increase the quality of our simulator. We

measure the current consumed by using the National Instrument USB-6210 with MATLAB (16-bit datum per minute). Table 4 shows the power consumption of the sensor node's main components when using a super-capacitor charged at 3V. For the sensing and transmission operations (e.g. *Read Light Sensor + BLE Transmission)*, the current includes all the sensing and communications steps such as the sensor reading, the advertisement phase, the connection, the transmission of the data using BLE, and the final acknowledgment message. Therefore, we include in this value all the energy consumed by the node during its wake-up time.

Table 4. Sensor-Node Current Features. For the sensing and transmission operations (e.g. *Read Light Sensor + BLE Transmission*), the current includes the sensor reading, the advertisement phase, the connection and transmission of the data using BLE and the final acknowledgment message.

| Feature | Current [μA] |
|---|---|
| Board Leakage + MCU in Sleep Mode | 3.5 |
| Read Light Sensor + BLE Transmission | 199 |
| Board + PIR and MCU in Sleep Mode | 4.5 |
| PIR Detection | 102 |
| Solar Panel at 200 lux | 31 |
| Solar Panel at 50 lux | 7.75 |

*4.1.2 Light Measurements:* We placed a node in different types of locations and measured light intensity, supercapacitor voltage at 5-minute intervals for 15 days. The locations are: *(i)* a windowless *Conference Room* where light is On only when people occupy the room; *(ii)* a *Staircase* where internal lights are always On for security reasons; *(iii)* the *Middle of an Office* room, mainly subject to internal lights; *(iv)* a node subject to natural light from a *Window*; and *(vi)* a node placed close to the *Door* of an office room where light intensity is low.

*4.1.3 Modeling the Energy Storage Level:* The super-capacitor accumulates energy when light is available and depletes energy with node operations.

*Energy Produced:*

$$E_{Prod} = Solar_{Power} * LightIntensity * TimePeriod \tag{7}$$

From the solar panel (i.e. AM-1454 for indoor usage) data-sheet [12], the current generation per 200 lux of light at 2.5V is 35.2 uA. Hence, the approximate energy generated by the solar panel at 200 lux in one second is 88e-6J (35.2uA * 2.5V * 1s). As we measure light intensity only once per 5 minutes, we miss light fluctuation events. We ignore solar plan inclination, the wavelength of light and reflection of lights to keep the model simple. Hence, the energy estimated is an approximation. However, we show that the model is sufficient to learn a policy that works in the real world.

*Energy Consumed:*

$$E_{Cons} = E_{Send} + E_{Sleep} \tag{8}$$

$E_{Send}$ is the energy consumed to read and transmit a sensor data packet, and $E_{Sleep}$ that is the energy consumed by the sensor node in between two transmissions.

*4.1.4 Validation of Modeling.* We validate our discharge model when no light is available. Figure 5 compares the lifetime of the sensor-nodes in simulations (black) and real data (red) using different sensing rates. The two trends capture the high-level characteristics well but show small differences due to real-world events that are difficult to model. The sensor-node lasts up to 9 hours by collecting data every 15 seconds, up to 34 hours at a 1-minute sending rate, and

up to 6.25 days with a 10-minute sending-rate. Hence, the RL agent needs to take the right action based on the current environment state to maximize the sensing rate while avoiding energy depletion. We validate our charging model with current measurements in different light conditions as shown in Table 4.

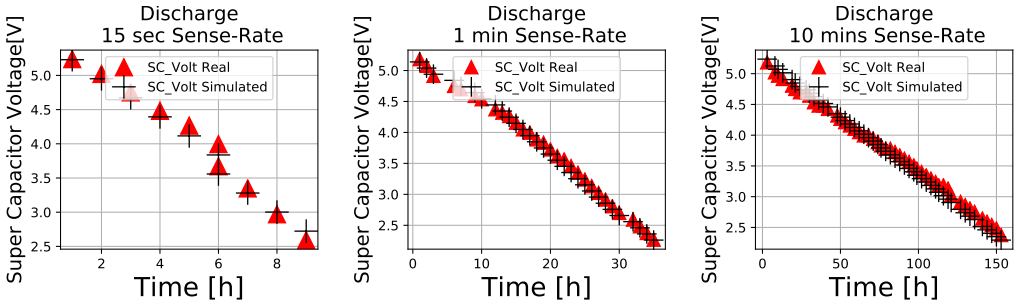


Fig. 5. Super-Capacitor Discharge Comparison between our Simulator and Real-World Using Different Sensing Rates

*4.1.5 RL Environment Setup.* The simulator interacts with the RL agent as explained in Algorithm 1. Equations 7 and 8 keep track of the energy voltage level and the light intensity is taken from real-world measurements. We model the environment state as follows:

*Light intensity*: We normalize the light intensity from a range of 0 to 10, where 0 represents no light and 10 represents 2000 lux or above. We select 2000 lux as a maximum value after checking typical indoor light intensity in buildings.

*Energy storage level*: We scale the energy storage voltage from 0 (min voltage 2.1V) to 10 (max voltage 5.5V).

*Weekend/Weekday*: Buildings' indoor lights patterns are strongly dependent on the presence of people [8]. Hence, we consider a binary state to capture weekdays and weekends.

Once the super-capacitor voltage reaches <2.1V, it terminates all its operations and energy recovery can take hours. To avoid long communication gaps between the sensor node and the base station, we penalize the RL agent with high negative reward (-300) when super-capacitor voltage is <3V.

### 4.2 Simulation Results

We present the results of policies learned after deploying 5 different nodes in 5 different lighting conditions. Each node collects light measurement data every 5 minutes for 15 consecutive days. We run 5 different simulations, one per node, with the respective data collected. Figure 6 shows the results of the simulation.

From Figure 6-left for the Conference Room, we see that ACES uses the lowest node sleep time (i.e. action 3) when lights are on and the energy storage is almost full (SC voltage level is 9), but as soon as lights turn off, it increases the node sleep time (i.e. action 0) to save energy. The conference room has no windows, has long periods of time with lights off and hence, light patterns are sporadic. The system learns that to avoid energy storage depletion, it is better to save energy as soon as lights turn off. When the lights turn on again and the energy levels are not full, ACES switches between action 2 and 3 to allow the energy-storage to recover to full charge.

From Figure 6-center for the Stairs Access, we notice that light is always on at level 1 (200 lux) due to security reasons, but the intensity is not enough to keep the super-capacitor charged. Hence, ACES uses a higher node sleep time that allows slower charging of the super-capacitor over time. When the voltage level reaches the maximum (level

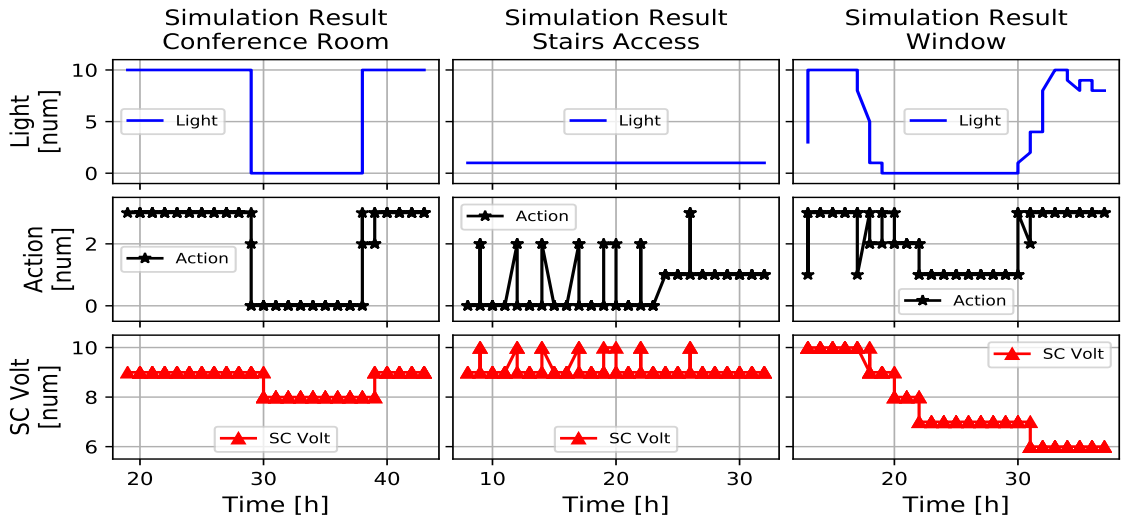


Fig. 6. Simulation Result on Different Lighting Conditions. Results obtained by running ACES using 15 days of lighting data traces.

10), ACES uses a lower node sleep time (action 2 or 3). That drops the super-capacitor voltage and forces ACES to use lower actions again.

From Figure 6-right for the Window, we notice that ACES uses the lowest node sleep time (action 3) when the light is on and energy-storage is full, but it starts using lower actions as soon as lights go off and the super-capacitor reduces its voltage. But compared to the conference room case, the node sleep time increase is gradual as ACES learns light patterns from sunset to sunrise. It switches to a lower node sleep time even when there are no lights, forecasting that the light will become available in the next few time-steps.

*4.2.1 Convergence Time.* For each simulation, we collect the total reward obtained at the end of each episode and average them to show the convergence of the algorithm over time. In Figure 7, we show an example of the average reward convergence while running a simulation using $\alpha$ equal to 0.1 in the Window location. The convergence happens around 12500 episodes. The entire simulation takes 30 minutes of wall-clock time using an Intel Core-i7 CPU with our Python implementation.

*4.2.2 Input/Action Space Analysis.* The dimension of the Q-Table is the product of the input and action state. By increasing the number of states: (i) we can represent the input and output variables more accurately, (ii) we need more iterations for the Q-learning algorithm to converge to a working solution. To better understand how this trade-off behaves in our system, Tables 5 and 6 show the rewards achieved while selecting different inputs and actions respectively.

Table 5 reports the reward achieved by ACES when the input uses (i) the only super-capacitor voltage (i.e. SC); (ii) The *SC* and *Light* measure; (iii) the *SC* and week/weekend day (i.e. *Week*); (iv) *SC, Light and Week* as for ACES and (v) *SC, Light, Week* and *Time* of the day that is expressed in hours. For these simulations, we use an action space equal to 4. Simulations and rewards are calculated on the same 15 days of light data as in Section 4.2 but only the first 7 days of data are used for training. Using the only supercapacitor as an input state gives high reward on places where light is low in intensity (Door, Stair). Introducing *Light* as an input state increases rewards where light has

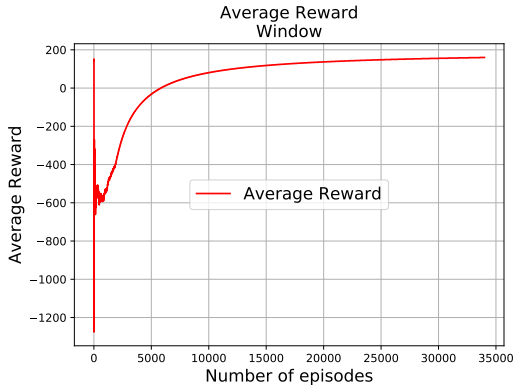


Fig. 7. Average Reward Simulation Results for Window using $\alpha$ equal to 0.1. It takes less than 30 minutes for our algorithm to converge on a typical Intel Core-i7 CPU.

Table 5. Input Space Analysis

| | Door [Rew] | Stair [Rew] | Middle [Rew] | Conf [Rew] | Window [Rew] | Avg Rew [Rew] | Input Size [num of states] |
|---|---|---|---|---|---|---|---|
| SC | 392* | 723 | 322 | 551 | 1310 | 670 | 11 |
| SC-Week | 179* | 723 | 322 | 353 | 1510 | 617 | 22 |
| SC-Light | 135 | 699 | 527 | 611 | 1270 | 648 | 121 |
| SC-Light Week | 161* | 721 | 832 | 1040 | 1413 | 833 | 242 |
| SC-Light Week-Time | 319 | 722 | <0 | <0 | 1304 | <0 | 5808 |
| SC = Super Capacitor Voltage; Time = hours of the day; Rew = Reward | | | | | | | |

high variability throughout the day (Conference and Window). As an average of the five lighting conditions, the use of *SC, Light and Week* is the one that brings more rewards to the system. The increase in rewards comes at the cost of having as an average an input space that is 22 times bigger (i.e. 242 in the Table) compared to using only the SC (i.e. 11). In the same Table, we conducted another experiment to understand why the reward achieved by the *SC* and *SC-Week* was higher than *SC-Light-Week* for the *Door* case: we run the same experiment for 2.5 months of data and noticed that *SC-Light-Week* achieves a higher reward in the long term (9371) w.r.t. SC-Week (9176) and SC (8186). With *SC-Light-Week*, ACES uses more information to converge to a better policy, but it requires more data as it needs to learn the policy for all variations that exist in the given observations. The use of the *SC-Light-Week-Time* as an input brings the system to achieve a negative reward in 2 places. This is due to a very large input state and hence ACES could not converge to a good solution within 24 hours for which we ran the simulator. For accommodating these larger state space, we would need to use function approximation algorithms such as Deep Q-Networks [44].

Table 6 shows the effect of changing the action space. Since the reward collected is equal to the action selected by the system at each step, for this experiment we normalized all the rewards from 0 to 3. We use *SC-Light-Week* as the input state. Increasing the number of actions increases the final reward on average. We use 4 actions in our real-world experiments to keep the Q-Table small.

Table 6. Action Space Analysis

| Reward | Door | Stair | Middle | Conf | Window | Avg |
|---|---|---|---|---|---|---|
| Action = 2 | 81 | 147 | 607 | 240 | 1359 | 486 |
| Action = 4 | 161 | 721 | 832 | 1040 | 1413 | 833 |
| Action = 8 | 435 | 1097 | 649 | 1235 | 1923 | 1068 |

*4.2.3 Event-Driven Applications.* The PIR sensor adds to the sleep current of the node by 1$\mu$A, each event consumes an additional 102 $\mu$A (Table 4) and lasts for 2.5 seconds. ACES needs to account for this additional energy use when selecting the sensing rate. We simulate stochastic events in the environment with an average of 50 events per day during weekdays and 20 events per day during the weekends, a conservative estimate of daily events as reported by Agarwal et al. [3]. For these simulations, we use the same 15 days of data lighting traces as for Figure 6, so that a comparison with previous results is fair. Again, each RL training uses lighting data collected by one node.

Table 7. ACES comparison between Periodic Sensing Applications and Event-Driven Applications. Results obtained after training and running ACES on the same 15 days of data lighting traces used for Figure 6.

| Node Placement | Periodic-Sense [avg duty-cycle period-period in sec] | Event-Driven [avg duty-cycle period-period in sec] | Percentage [%] |
|---|---|---|---|
| Conference Room | 39 | 40 | 98 |
| Window | 30 | 31 | 96 |
| Middle | 135 | 140 | 97 |
| Door | 128 | 171 | 72 |
| Stairs Access | 81 | 95 | 85 |

Table 7 compares the data-packets sent by the final policy with and without the PIR sensor. In all the scenarios, ACES successfully accounts for additional energy expenditure from event-driven sensors and continues to maintain perpetual operation. When abundant light is available throughout the day (i.e. windows, conference room and center of office), the number of data packets sent is similar to the baseline periodic sensing (95% to 98%). However, in stair access and door the light availability is low, and hence, the number of data samples drops to 85% and 72% respectively.

*4.2.4 Complexity of the Proposed Methods.* In Q-learning with $\epsilon$-greedy exploration, the convergence time of the algorithm has an upper bound of O(exp(number of states)). To keep the training time tractable, we discretize the state/action space and keep the number of states to a minimum. We limit our training time to 30 minutes with a single CPU. We can scale to more number of states using deep RL techniques such as DQN [44], which we leave for future work. ACES uses three variants of the Q-learning algorithm: one-time learning, day-by-day learning and transfer learning. The number of states do not change between these algorithms, hence the complexity of the algorithm remains the same as Q-learning. All three versions are trained with our simulator until Q-values stabilize or 30 minutes elapse. The primary difference between these algorithms is the amount of external lighting data made available to the simulator, which in turn affects the states that are given as input to the agent. The one-time learning policy is trained only once with available data, while the day-by-day and transfer learning algorithms are trained periodically every day. The first policy trained takes as much time to converge as one-time learning. However, subsequent training converge faster as the data collected captures all the scenarios in the environment and Q-value estimates stabilize.

Figure 8 shows the convergence of the Q-Table values for one-time and transfer learning. The graphs are averaged for the actions. Therefore, we have 4 curves, one for each action. It takes 12K episodes for Q-values to stabilize in one-time learning. The transfer learning policy takes only 3K episodes to converge as it starts training from a general policy trained with data from multiple sensor nodes.

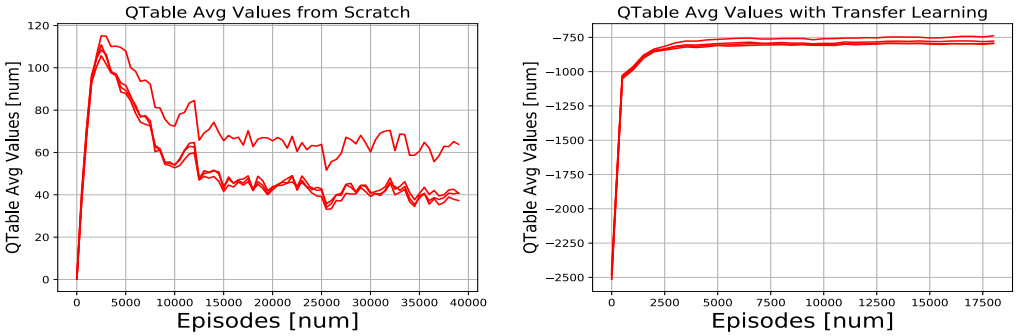


Fig. 8. Convergence of the Q-Table values for one-time and transfer learning. The graphs are averaged for the actions. Therefore, we have 4 curves, one for each action. It takes 12K episodes for Q-values to stabilize in one-time learning. The transfer learning policy takes only 3K steps to converge as it starts training from a general policy trained with data from multiple sensor nodes.

## 5 REAL-WORLD EXPERIMENTAL RESULTS

### 5.1 One-Time Learning

We learn policies using the simulator. Each policy trains by using 15 days of light data collected from the respective node. Then we use the resulting Q-Tables to control the real sensor nodes in the respective locations for 31 days. We continue to update the policies based on real-world data while using $\epsilon = 0.9$. The Q-learning updates use the parameters reported in Table 8 that also indicates the training-time phase and deployment-time phase.

Table 8. Q-Learning Hyper-parameters for One-Time learning experiment on Real-World. The only parameter that changes compared to the simulation experiment is the $\epsilon$ that here is fixed and set to 0.9.

| Hyper-Parameter | Value | Parameter | Value |
|---|---|---|---|
| Reward-decay ($\gamma$) | 0.99 | Observations | State of Charge, Light Intensity, Week/Weekend |
| Epsilon-fixed ($\epsilon$) | 0.9 | Actions | 15s, 60s, 300s, 600s |
| Learning rate ($\alpha$) | 0.1 | Node "Death" Threshold | 3V |
| Wait time $T$ | 15 mins | Q-Table Convergence | <5% change in mean Q value |
| Episode Duration | 24 hours | Training-phase | 15 days |
| | | Deployment-phase | 31 days |

Notice that the only parameter change compared to the simulation experiment is the $\epsilon$ that now is fixed and set to 0.9. Thus, the agent takes the majority of the actions based on learned Q-values from the simulator but minimally takes

random actions to learn changing patterns. If the agent encounters a state not listed in the Q-table, it takes a random action.

Figure 9 shows the results of three of the five nodes. The behavior in real deployment is very similar to the simulation results. There are several outliers in the sequence of actions due to $\epsilon$-greedy exploration. In the *Door* location (Figure 9-left), the light intensity is low during the day (it reaches level 3 at most) and ACES gradually selects higher actions when enough light is available. For the Stair Access case (Figure 9-middle), the light availability is low, hence ACES picks lower actions (i.e. 0 or 1).

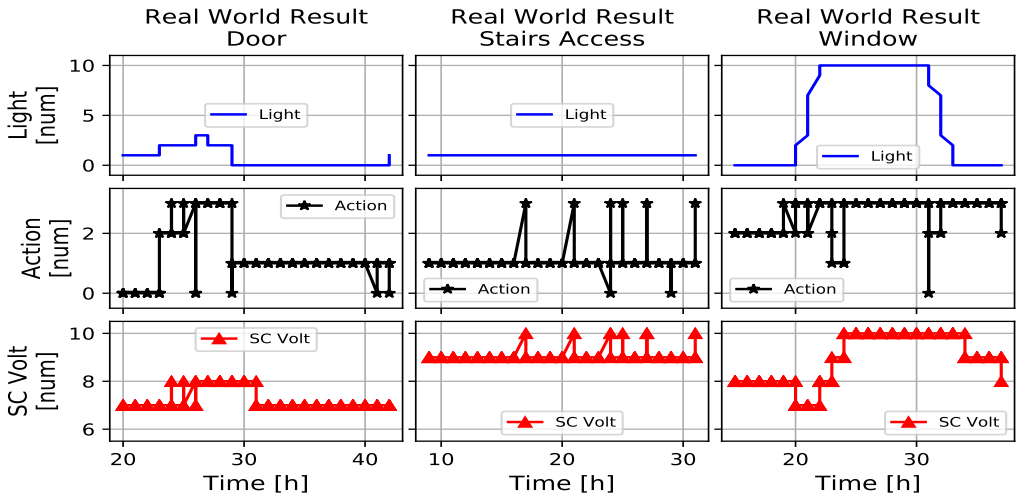


Fig. 9. ACES Real-World Results in Different Lighting Conditions. As a first experiment, we use the ACES learned policies (i.e. Q-Tables) from the simulator to drive the same five nodes used to collect the 15 days lighting data traces. We leave ACES driving the nodes for 31 days. We use $\epsilon$ equal to 0.9 to leave the system exploring in the real-world. This is why we can see several outliers in the middle of a constant sequence of actions.

For the Window location (Figure 9-right), ACES uses the lowest duty-cycle period (action 3) during the day and decreases the action to 2 when the light goes down. However, the action is never lowered to action index 1 as observed in simulations because the energy-storage level never drops to <6. Upon further digging, we found that the communication between the sensor node and the base station on average took more time than in simulation. Hence, data is exchanged less frequently (every 17 seconds instead of 15) and the power used is lower as well. The policy automatically adjusted to this difference in the environment to maximize its rewards.

The *Center of Office* results is similar to the *Door* node results since it has similar light patterns. The *Conference Room* node also performs close to the simulation results.

*5.1.1 Comparison with Fixed Periodic Sensing.* Table 9 compares the data sent by ACES with a fixed sensing period of 1 minute used commonly in buildings. ACES outperforms the average duty-cycle period compared to a fixed duty-cycle period when there is a consistent amount of light as the agent learns the daylight patterns and uses a duty-cycle period of 15 seconds when the light is available (*Window* case). The average duty-cycle period is similar (96%) for the node placed in the *Conference Room*. This value is closely related to the amount of light available and the presence of people in the environment. Percentages are lower for *Center of Office*, *Door* and *Stairs Access* as the light available is lower.

Table 9. Placement-QoS comparison between a battery-powered system and ACES

| Node Placement | Battery-Power [avg duty-cycle period in sec] | ACES [avg duty-cycle period in sec] | Percent [%] | Dead Time (%) |
|---|---|---|---|---|
| Conf. Room | 60 | 62 | 96 | 0 |
| Window | 60 | 35 | 170 | 0 |
| Center Office | 60 | 82 | 73 | 0 |
| Door | 60 | 136 | 44 | 0 |
| Stairs Access | 60 | 122 | 49 | 0 |

Even for these locations, sensor nodes send data every ~ 2 minutes (Stairs Access: 49%, Door: 44%). All the 5 nodes avoided battery depletion and have 0% dead time throughout the 31-day real-world experiment.

### 5.2 Day-by-Day Learning

Instead of learning from the simulator just once with several days of data, we switch to learning a new policy from the simulator every day. Each policy is learned by using lighting data traces of the respective node. For this experiment, we deploy five nodes in five different lighting conditions for 15 days. In our One-Time Learning deployment, we observed that by leaving the RL to explore (i.e. $\epsilon$ = 0.9) a new sequence of actions can decrease the quality of service of applications when it takes a random action once in a while. Hence, it is better to avoid exploration while running the sensor node in the real world.

On the first day, we start with a fixed policy of collecting data every 15 mins. At the end of the day, the collected data is used to learn a new Q-table in the simulator by running Algorithm 1 with the hyper-parameters reported in Table 3. The learned Q-Table is used to collect data the next day with $\epsilon$ set to 1. The hyper-parameters used to collect the data for day-by-day learning are presented in Table 10.

Table 10. Q-Learning Hyper-parameters for Day-by-Day learning experiment on the Real-World. To avoid exploration in the real world, we set and maintain the $\epsilon$ value to 1.

| Hyper-Parameter | Value | Parameter | Value |
|---|---|---|---|
| Reward-decay ($\gamma$) | 0.99 | Observations | State of Charge, Light Intensity, Week/Weekend |
| Epsilon max ($\epsilon_{max}$) (For Simulation) | 1 | Actions | 15s,60s, 300s, 600s |
| Epsilon min ($\epsilon_{min}$) (For Simulation) | 0.1 | Node "Death" Threshold | 3V |
| Epsilon-fixed ($\epsilon$) (During Deployment) | 1 | Q-Table Convergence | <5% change in mean Q value |
| Learning rate ($\alpha$) | 0.1 | Episode Duration | 24 hours |
| Epsilon increment ($\Delta$) | 0.0004 | Training-phase | repeat every 24 h |
| Wait time $T$ | 15 mins | Deployment-phase | 15 days |

The only hyper-parameter that changes when the system takes actions in the real-world compared to the simulation experiment is the $\epsilon$ that is fixed and set to 1 during deployment.

From the second day, the training starts from the previous day's Q-table that summarizes the learning until that day. Thus, the Q-Table updates day-by-day from the first day of the deployment. For this experiment, we leave ACES driving five nodes in five different lighting conditions for 15 days.

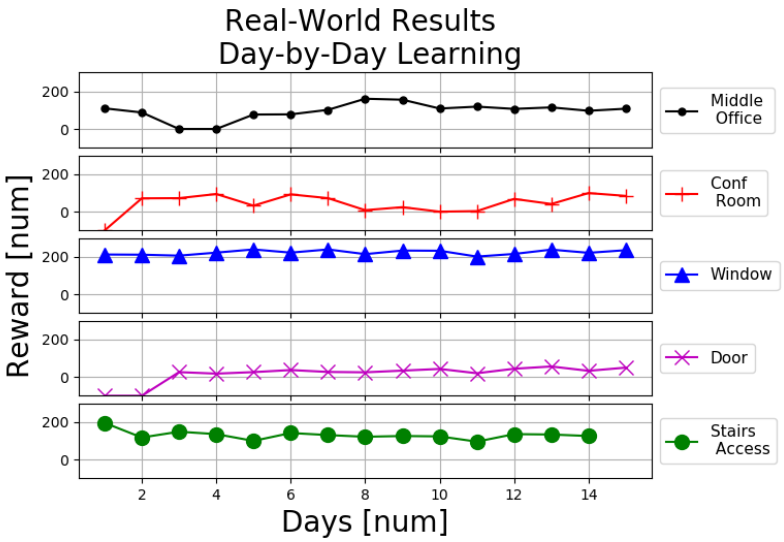


Fig. 10. ACES Real-World Results in Different Lighting Conditions using Day-by-Day learning: instead of learning from the simulator just once with several days of data (i.e. 15 days in our previous experiment), we switch to learning a new policy from simulator every day. For this experiment, we leave ACES driving five nodes in five different lighting conditions for 15 days.

Figure 10 shows the rewards obtained by the five nodes placed on the five different lighting conditions for 15 days. The figure cuts off with a value of -10 which indicates the nodes died because of energy storage depletion. 3 out of the 5 nodes deployed (i.e. *Middle Office*, *Window* and *Stair Access*) had 0% dead time while two nodes (i.e. *Conference Room* and *Door*) depleted their energy in the first days of the experiments. The lighting energy in the latter case is mainly subjected to human behavior (i.e. turning on lights or blinds) that can change between days. ACES needs some time to understand those patterns and adapt to them. On the other hand, the nodes that maintain continuous operations are the ones that are mainly subject to a constant light pattern - light is available from sunrise to sunset for the Window node and the light is always on for security reasons near the Stair Access node. For this experiment, we experience a dead time of 4%.

To increase the validation of *Day-by-Day* learning, we further deployed 10 more nodes for 15 days. We upgrade our boards to use a 1.5F super-capacitor to extend node lifetime in case there is no light available. The nodes achieve 0% dead time in 15 days of deployment.

### 5.3 Transfer-Learning

We speed up the learning process for a multi-node deployment with transfer learning [56].

*5.3.1 General Q-Table.* We use a week of lighting data collected for the day-by-day experiment (Section 5.2) for all the five different lighting conditions (a total of 5 weeks of light data) to run a single ACES simulator and build a general Q-Table. Then, we place five new sensor nodes in similar lighting conditions as the previous nodes, i.e. a node close to a

window, a node in another conference room and so on. We run ACES on the new nodes while starting them with the general Q-Table instead of learning everything from scratch.

Table 11. ACES results after adopting transfer learning after the first 3 days of deployment. Between parentheses, we show the number of times the nodes died during deployment.

| Avg Duty-Cycle Period in sec (Num Node Died) | Empty Q-Table | General Q-Table | Percentage [%] |
|---|---|---|---|
| Conference Room | 98 (1) | 64 (0) | 151 |
| *Avg Light [lux]* | *520* | *455* | *89* |
| Window | 25 (0) | 26 (0) | 98 |
| *Avg Light [lux]* | *4518* | *3888* | *87* |
| Middle Office | 101 (0) | 90 (0) | 108 |
| *Avg Light [lux]* | *281* | *340* | *121* |
| Door | 348 (2) | 301 (1) | 115 |
| *Avg Light [lux]* | *117* | *99* | *85* |
| Stairs Access | 90 (0) | 93 (0) | 97 |
| *Avg Light [lux]* | *184* | *181* | *98* |

Table 11 reports the average duty-cycle period in seconds for the five different places for the first 3 days of deployment by starting ACES using a general Q-Table versus an empty one. Between parenthesis, we report the number of times the nodes died. By using a transfer learning approach, the nodes perform better as indicated by a decrease in the average duty-cycle period. The nodes subject to easy to predict patterns (*Stair Access* and *Window*) have similar results regardless of an empty table or a general table, ACES can find the best sequence of actions after only one day. The situation is different for the placements subject to human occupancy patterns (*Conference Room*, *Middle Office*, and *Door*): in these cases, the general Q-Table helps speed up the learning process. The node placed in the conference room sent up to 1.5x more data. Most importantly, the general Q-Table helps the nodes to reduce dead time: in case of the conference room, this is reduced from 1 to 0, while in the Door case this is reduced from 2 to 1.

*5.3.2 Similar Lighting Q-Table.* We also tried transfer learning by starting 5 different nodes while using a Q-Table already generated by nodes running in similar lighting conditions (e.g. *Window* to *Window*). All the nodes benefited from the initial policy and achieved 0% dead time across 2 weeks of deployment.

### 5.4 Comparison with State of the Art Solutions

We compare the average duty-cycle period for periodic sensors used by ACES using a day-by-day learning policy with *(i)* an energy manager based on reinforcement learning for energy harvesting wireless sensor networks (RLMan [4]), *(ii)* a rule based power management algorithm that was tuned to maximize sensing-rate while avoiding energy depletion on energy-harvesting battery-less motes (i.e. Mote-Local [21]), *(iii)* a reinforcement Q learning-based throughput on-demand provisioning dynamic power management method (RLTDPM) for sustaining perpetual operation and satisfying the ToD requirements for energy harvesting wireless sensor node [30], *(iv)* a dynamic sampling rate adaptation scheme based on reinforcement learning (Q-learning), able to tune sensor sampling interval on-the-fly (On-Line RL), according to environmental conditions and application requirements [15] and *(v)* a battery-powered system that send data every minute [43]. We consider these 5 architectures since they include current solutions (*v*), literature heuristic (*ii*) and RL-based methods (*i, iii, iv*).

Table 12. Average duty-cycle period (lower is better) comparison between different methods w.r.t. ACES on different indoor lighting conditions.

| Placement [Avg Duty-Cycle Period in sec] | Window | Conf. Room | Middle Office | Door | Stairs Access | Avg Without Door | ACES Improvement |
|---|---|---|---|---|---|---|---|
| Battery-Powered Baseline [43] | 60 | 60 | 60 | 60 | 60 | 60 | + 18 % |
| Mote-Local [21] | 57 | 149 | 128 | 337 | 94 | 107 | - 34% |
| RLMan [4] | 30 | 133 | 150 | 411 | 136 | 112 | - 37% |
| On-Line RL [15] | 69 | 165 | 103 | / | 98 | 109 | - 35% |
| RLTDPM [30] | 30 | 124 | 117 | / | 129 | 100 | - 29% |
| ACES | 24 | 123 | 99 | 328 | 63 | 71 | |
| / = node died during experiment | | | | | | | |

RLMan uses an actor-critic algorithm with function approximations and learns a policy based on historical data with only the energy storage level as states. We use the same state, but use the Q-learning algorithm to learn a policy. We run the algorithm until it converges in the simulator, hence the change in algorithm will not affect the final policy learned. To simulate the Mote-Local method, we used the power management algorithm described in [21]: the system increases the duty-cycle period if the lighting is available and the supercapacitor is increasing the voltage with time and decreases the sensing-rate when the light is off or the voltage is decreasing or maintaining the same value with time.

For this experiment, we use a week of lighting data collected for the day-by-day experiment (i.e. Section 5.2). RLTDPM is a one-time learning method. To compare it against ACES we limited the learning of the environment from the simulator to the first half of the week and let the agent take actions in the remaining half. In [15] (i.e. On-Line RL), the system can only take three actions: increase, decrease or maintain the current duty-cycle period. Therefore, we modified the simulator accordingly. On-Line RL [15] also uses one-time learning. Therefore, in a week-long experiment, we considered On-Line RL training using only the first half of the week data.

Table 12 shows the comparison for each lighting condition.

ACES outperforms the average duty-cycle period compared to the RLMan, RLTDPM, On-Line RL, and Mote-Local techniques. RL beats the Mote-Local technique as it learns the impact of each action it takes with rewards received, the formulation teaches the agent to forecast conditions based on historical data and optimize for lower duty-cycle periods. ACES beats RLMan because it uses additional state information such as light intensity and weekday/weekend. Hence, these additional states we include in ACES makes a measurable impact on the performance of the node.

From the Stairs Access case, the duty cycle that RLTDPM can achieve is only 129 sec while ACES reaches 63 seconds. Our system obtains better rewards because of day-by-day learning. Therefore, ACES continuously learns from the environment and adapts its performance over-time even in critical low lighting conditions (e.g. Door). We put a '/' in the Door case since the node died during the simulation. On-Line RL reaches larger duty-cycle periods than ACES in every condition due to the incapacity of the system in selecting an intermediate duty cycling period when needed (i.e. Action 1 and 2). On-Line RL also depletes its energy storage in the Door case due to its inability to consider changes in the environment over-time. Our system avoids energy storage depletion thanks to day-by-day that reduces dead time to near 0.

### 5.5 Real-World PIR Event + Periodical Sensing

We deployed 45 nodes with a PIR sensor to evaluate ACES in the real world for two weeks. We place 9 nodes for each of the five lighting conditions. The nodes send both PIR events and light intensity data. 40 nodes use day-by-day learning policy and 5 nodes use transfer-learning policy. As event sensing nodes remain awake much of the time, we increased the super-capacitor size to 1.5F to account for additional energy expenditure. A node now lasts up to 9.6 hours when it sends a packet every 15 seconds with the PIR always on. It lasts up to 8 days with 10 minutes sensing period. Table 13 reports a summary of the nodes deployed in the different lighting conditions, including the number of times the nodes die and the time in hours that they were off. For the *Door* case, we placed two of the 9 nodes close to the ceiling near a source of light to facilitate the detection of people entering or leaving the room, and hence the average light the nodes achieves in this location is higher than other averages.

Table 13. A summary of the 45 nodes we deployed for PIR event-detection. 9 nodes deployed for each lighting condition. Numbers are average per day during a two weeks experiment.

| Node Placement | Avg Light [lux] | Avg PIR [event] | Peak PIR [event] | Avg Sensing [sec] | Dead Time [h] | Node Dead [num] |
|---|---|---|---|---|---|---|
| Conference | 1139 | 43 | 83 | 119 | 24 | 1 |
| Window | 4301 | 83 | 199 | 79 | 0 | 0 |
| Middle | 423 | 61 | 175 | 107 | 0 | 2† |
| Door | 554* | 33 | 85 | 110 | 0 | 0 |
| Stairs/ Corridors | 479 | 81 | 154 | 149 | 0 | 1† |
| * 2 nodes out of 9 in the ceiling near internal lights | | | | | | |
| † 3 nodes died in the *Middle* and *Corridor* cases due to hardware defects | | | | | | |

3 nodes died in the *Middle* and *Corridor* cases. Upon investigation, we found that the nodes were defective and did not charge even when lights were on. For the *Conference* case, most nodes performed well, but one of the nodes ran out of energy for about 24 hours. Upon checking its historical data, we found that the light in the room remained off for several days and the nodes died during the weekend. After people entered the room and turned on the light, the node resumed operations in just 15 minutes. Results are positive for the *Stairs/Corridor* case where nodes detect as an average of 81 events per day and sent light data every 149 seconds. The *Window* location is the most performant since sensors can send data on average every 79 seconds with 199 motion events captured.

*5.5.1 PIR Detection Accuracy.* To evaluate how many events are missed by ACES, we placed 15 battery-powered nodes as ground truth for the events detected. We count at most one motion event every 2 minutes for both the ACES and ground truth sensor nodes. Table 14 compares the events detected in the different lighting conditions w.r.t to the ground truth nodes. The Table reports the average events per day. The Table shows that for the *Window* case the number of data sent from ACES w.r.t to a battery-powered node is 99%. This is not surprising as the majority of events happen during the day. *Conference* nodes also get 91 % of the events since as soon as people enter the room, the super-capacitor gets fully charged in a few minutes. The *Corridor* case is the most challenging, where there is a continuous stream of events due to people moving and ACES catches only 73% of them (mean of 112 event per day). This can be mitigated with a different event detection strategy, e.g. reward events detected every 10 mins, for locations with large activity.

Table 14. Event detection comparison between ACES and ground-truth nodes. Events averaged per day.

| Node Placement | Ground-Truth [events] | ACES [events] | Percentage [%] |
|---|---|---|---|
| Conference Room | 52 | 48 | 91 |
| Window | 110 | 109 | 99 |
| Middle | 125 | 98 | 79 |
| Door | 63 | 54 | 86 |
| Stairs/Corridors | 154 | 112 | 73 |

*5.5.2 Continuous Operation and Morning First Event Detection.* Barring the 3 defective sensors, 41 nodes out of the 42 deployed maintained continuous working operations. More importantly, they were operational throughout the night and detected the first PIR event in the morning when people come into their office. The first morning PIR event is important for convenience as users expect a fast response from building systems when they enter their office. ACES detects 99% of these events and only one was delayed by 15 minutes.

*5.5.3 Transfer Learning.* We deployed 5 of the nodes using transfer learning with a Q-Table from a node in a similar lighting condition described in Section 5.3.2. All the nodes benefited from the initial policy and achieved 0% dead time.

### 5.6 Energy Neutral Operations

It is possible that the ACES nodes perform well in our multi-week deployments, but are consuming incrementally more energy than is available and die out in a longer deployment. We evaluate the energy neutrality of ACES nodes by monitoring the super-capacitor voltage level of five randomly picked nodes in each type of location. If the super-capacitor voltage level steadily decreases over time, ACES nodes will die more frequently in longer deployments. Table 15 shows the super-capacitor voltage percentage at midnight after two consecutive days for the five nodes.

Table 15. Energy neutral operations after consecutive days for different lighting conditions

| Node Placement | Midnight Day1 [SC Volt in %] | Midnight Day2 [SC Volt in %] | Difference [num] |
|---|---|---|---|
| Conference Room | 93 | 100 | 7 |
| Window | 88 | 87 | 1 |
| Middle | 88 | 87 | 1 |
| Door | 67 | 68 | 1 |
| Stairs/Corridors | 91 | 91 | 0 |

For 4 out of the 5 nodes, the voltage level in the super-capacitor stays within 1% after 24 hours. All four nodes have voltage levels at <100%. Thus, the ACES agent learns to modulate its duty-cycle period such that it neither expends too much energy nor is it saving too much energy. Figure 11 shows energy-neutral operations for the *Stairs* Case. In this case, the light is almost constant throughout the day, and ACES adapts the action accordingly to maintain a constant voltage level. At the end of two consecutive days, the super-capacitor voltage level percentage remains the same.

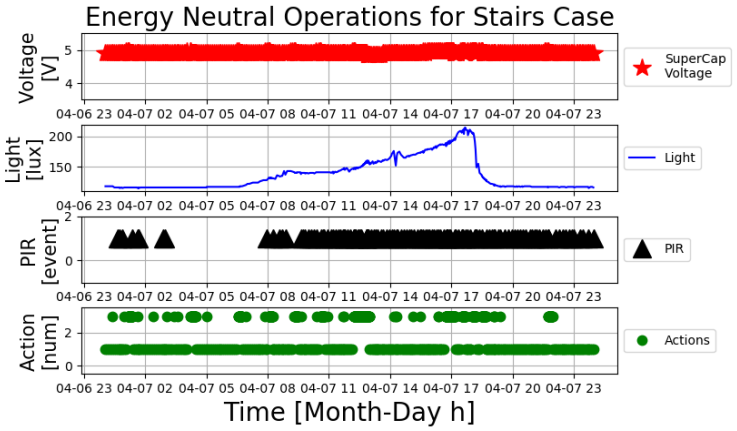


Fig. 11. Energy Neutral Operations for Stairs Case. The light is almost constant throughout the day, and ACES adapts the actions to maintain a constant voltage level. At the end of two consecutive days, the supercapacitor voltage level percentage remains the same.

For the Conference case, the situation is different and the supercapacitor voltage percentage reaches 100%. In this case, the ACES agent was conservative in spending its energy and did not send as much data as it could have. The conservative strategy may work better in the Conference room because of its unpredictable changes in light conditions.

## 6 REAL-WORLD EXPERIENCE AND LIMITATIONS

We have demonstrated that ACES can successfully set the sensing rate of energy harvesting devices according to light availability. Our simulation and real deployment results are promising. Our current deployment consists of 60 nodes across our department floor building with 15 nodes for periodical light sensing and 45 for PIR event-detection. We use 15 additional battery-powered nodes as a ground truth reference for the PIR nodes. Figure 12 shows the floor map and the position of the nodes. Based on our experience, we highlight the pertinent future research directions for using reinforcement learning at the edge.

*RL Problem Formulation:* The key to the success of reinforcement learning is to formulate the states, actions, rewards and state transitions to capture the essence of the problem. We empirically tried several formulations before finalizing the current design. This is a one-time effort that can be applied to the deployment of numerous sensors and is much better than manually configuring each sensor. However, tools to assist domain experts in the problem specification will be immensely useful to adopt ACES like solutions.

*Safe Exploration in Real-World Deployments:* In our one-time learning policy, we kept $\epsilon = 0.9$ to let the system explore during the day. Sometimes the randomly chosen action can drastically reduce the QoS. We avoided exploration by using the day-by-day learning strategy, where we learn the policy in the simulator. We can do this because of the effect of our actions on the environment were relatively easy to simulate. If that is not the case, we need a safe way to explore the real world deployment [6].

*Accommodate Larger State Space:* We carefully designed our states and actions to limit the search space of the Q-learning algorithm and make the problem tractable. However, in realistic IoT deployments where nodes can have tens of parameters, we may need to accommodate large state spaces and action choices. Recently proposed deep reinforcement learning algorithms such as DQN [44] and TRPO [51] can accommodate such large state spaces. As our agent resides in

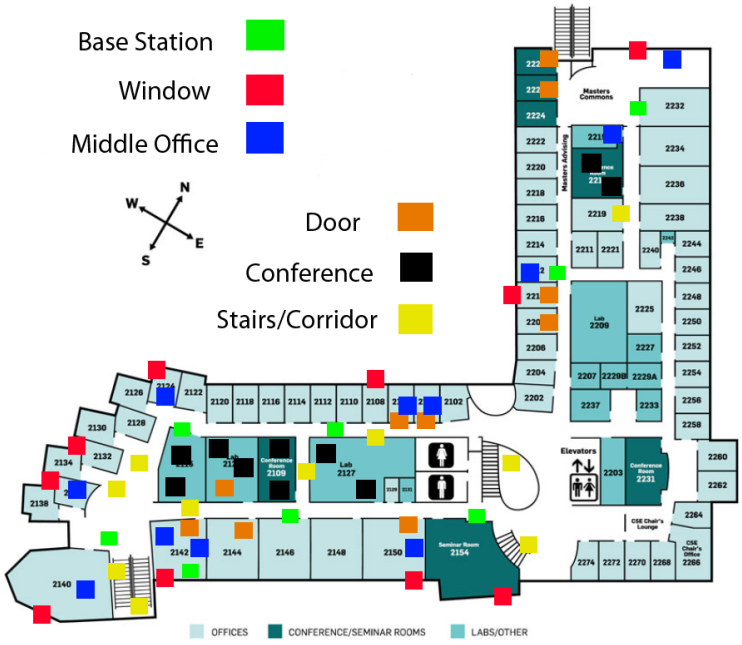


Fig. 12. 60 nodes deployed across our department building floor. 45 nodes send both PIR events and periodic light measurements, 15 nodes only sense light periodically. All the nodes use day-by-day learning and 10 nodes were initialized with transfer learning policy.

the power socket connected base station, it has enough computation power to deploy these algorithms. In future work, we plan to explore the computation versus performance tradeoffs among these algorithms.

*Speeding-up Learning:* In our current setup, we initially learned a new policy for each sensor node from scratch. To scale to thousands of sensors, we can learn a single policy that generalizes to many different contexts. We showed how transfer-learning can be used to speed up the learning process. However, we can learn a single policy by expanding the state space to include contextual features so that the RL agent learns the actions that maximize the long-term rewards in a context-specific manner. For our use case, we could add different states to include different information such as the type of room (conference room vs lobby), if the room has a window, etc. A single policy can learn from data collected by all the sensor nodes and hence will also speed up learning significantly.

*Managing Network Failures:* Communication between the nodes, the base station, and the computational unit can fail in the real-world environment due to unexpected events. During our deployment, our department network was subject to IP address reconfiguration of IP addresses by the network building manager and the local server running ACES failed to communicate the actions to the base stations and hence to the nodes. 3 nodes died during a 4-day disconnection. To limit network failures consequences, the Q-learning policy can be executed inside the node. The memory and compute requirements are low. The CC2650 MODA that is included in the board has 128Kb of programmable flash and the Q-Tables we learned are at most 25kb. As a proof of concept, we implemented a full Q-table inside one node and the node could modulate its sensing period without the need of an external computational unit. Note that the node's computational capability is not enough for our RL training with the simulator. However, we can recover from network failures by running the Q-Table from the sensor node itself as it only takes up to 25 Kbytes of memory.

*Simulation and Real World Differences:* While in the simulations the best sequence of actions is learned by considering all the nodes behaving identically, during our real-world deployment we discovered several factors that can differ between the nodes while severely impact the node behavior. We report them here as follows: (i) current leakage variability (ii) environmental interference, distance from the base station causing disconnections and more power needed for the node to reconnect. Taking into account those differences can increase nodes' performance. Furthermore, those differences become non-negligible in a large scale deployment. When we increased our deployment from 5 to 60 nodes, the disconnections in the network increased at a rapid rate. On average, 1 out of 20 packets is affected by disconnections and the node needs to reconnect with the base station causing increased energy consumption.

*Placement of Nodes:* One node never reached normal working functionalities because the light available was never enough for the node to charge. The node was placed in a corridor and only 60 lux was available during the day. After moving it closer to the source of light (110 lux on average), the node started working as expected. Hence, the placement of nodes should take into consideration the minimal energy required to operate the node.

## 7 CONCLUSION

Battery replacement of wireless sensors is a major bottleneck towards the adoption of large-scale deployments in buildings. We show that energy harvesting sensors provide a credible alternative when we optimize their operations using reinforcement learning (RL). Our system ACES uses RL to configure the sampling rate of solar panel based energy harvesting sensor nodes. Our sensor node uses Bluetooth Low Energy for communicating data to a nearby base station and senses both periodic measurements such as temperature and event-driven ones such as motion. Using both simulations and real-world deployments, we show that our Q-learning algorithm based RL policy learns to adapt to varying lighting conditions and sends sensor data across nights and weekends without depleting available energy. We explored deploying a one-time policy learned from historical data as well as learning a new policy based on data collected each day. Both of these strategies achieved perpetual operations in real-world deployments. We also show that transfer learning can be used to effectively reduce the initial percentage of node dead time in real deployments.

## ACKNOWLEDGMENTS

The research reported in this paper was sponsored by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

## REFERENCES

[1] acurite. '18. https://www.acurite.com/temperature-and-humidity-sensor-06002rm-592txr.html. ('18).
[2] adestotech.com. 2020. Home Page. https://www.adestotech.com/. (2020).
[3] Yuvraj Agarwal, Bharathan Balaji, Rajesh Gupta, Jacob Lyles, Michael Wei, and Thomas Weng. 2010. Occupancy-driven Energy Management for Smart Building Automation. In *Proceedings of the 2Nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building (BuildSys '10)*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/1878431.1878433
[4] Fayçal Ait Aoudia, Matthieu Gautier, and Olivier Berder. 2018. RLMan: an Energy Manager Based on Reinforcement Learning for Energy Harvesting Wireless Sensor Networks. *IEEE Transactions on Green Communications and Networking* 2, 2 (2018), 408–417.
[5] S. F. Barrett and D. J. Pack. 2006. *Microcontrollers Fundamentals for Engineers and Scientists*.
[6] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. 2017. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*. 908–918.
[7] Steven Bohez, Abbas Abdolmaleki, Michael Neunert, Jonas Buchli, Nicolas Heess, and Raia Hadsell. 2019. Value constrained model-free continuous control. *arXiv preprint arXiv:1902.04623* (2019).
[8] Bradford Campbell, Joshua Adkins, and Prabal Dutta. 2016. Cinamin: A Perpetual and Nearly Invisible BLE Beacon. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks (EWSN '16)*. Junction Publishing, USA, 331–332.

[9] Bradford Campbell and Prabal Dutta. 2014. An Energy-harvesting Sensor Architecture and Toolkit for Building Monitoring and Event Detection. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings (BuildSys '14)*. New York, NY, USA, 100–109.

[10] Bradford Campbell, Branden Ghena, and Prabal Dutta. 2014. Energy-harvesting thermoelectric sensing for unobtrusive water and appliance metering. In *Proceedings of the 2nd International Workshop on Energy Neutral Sensing Systems*. ACM, 7–12.

[11] Qingping Chi, Hairong Yan, Chuan Zhang, Zhibo Pang, and Li Da Xu. 2014. A reconfigurable smart sensor interface for industrial WSN in IoT environment. *IEEE transactions on industrial informatics* 10, 2 (2014), 1417–1425.

[12] Sanio Semiconductor CO. 2007. Amorphous Silicon Solar Cells. https://panasonic.co.jp/ls/psam/en/products/pdf/Catalog_Amorton_ENG.pdf. (2007).

[13] Haipeng Dai, Xiaobing Wu, Lijie Xu, and Guihai Chen. 2013. Practical scheduling for stochastic event capture in wireless rechargeable sensor networks. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 986–991.

[14] Samuel DeBruin, Bradford Campbell, and Prabal Dutta. 2013. Monjolo: An Energy-harvesting Energy Meter Architecture. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*. ACM, New York, NY, USA, Article 18, 14 pages. https://doi.org/10.1145/2517351.2517363

[15] Gabriel Martins Dias, Maddalena Nurchis, and Boris Bellalta. 2016. Adapting sampling interval of sensor networks using on-line reinforcement learning. In *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*. IEEE, 460–465.

[16] Joshua F Ensworth and Matthew S Reynolds. 2017. Ble-backscatter: ultralow-power IoT nodes compatible with bluetooth 4.0 low energy (BLE) smartphones and tablets. *IEEE Transactions on Microwave Theory and Techniques* 65, 9 (2017), 3360–3368.

[17] Khan Azam et al. 2011. Big data from the built environment. In *Proceedings of the 2nd international workshop on Research in the large*. ACM.

[18] S Mitchell Finnigan, AK Clear, Geremy Farr-Wharton, Kim Ladha, and Rob Comber. 2017. Augmenting Audits: Exploring the Role of Sensor Toolkits in Sustainable Buildings Management. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 2 (2017), 10.

[19] Cypress Envirosystems. 2015. Retrofitting Existing Buildings for Demand Response and Energy Efficiency. 2015. https://www.cypressenvirosystems.com/files/pdf/Retrofitting_Existing_Commercial_BuildingsV5.pdf. (2015).

[20] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, Luca Benini, and Rajesh Gupta. 2018. Pible: battery-free mote for perpetual indoor BLE applications: demo abstract. In *Proceedings of the 5th Conference on Systems for Built Environments*. ACM, 184–185.

[21] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, Luca Benini, and Rajesh K. Gupta. 2018. Pible: Battery-Free Mote for Perpetual Indoor BLE Applications. In *Proceedings of the 5th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building (BuildSys '18)*. ACM.

[22] Francesco Fraternali, Bharathan Balaji, and Rajesh Gupta. 2018. Scaling Configuration of Energy Harvesting Sensors with Reinforcement Learning. In *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems (ENSsys '18)*. ACM, New York, NY, USA, 7–13. https://doi.org/10.1145/3279755.3279760

[23] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 6 (2012), 1291–1307.

[24] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29, 7 (2013), 1645 – 1660. https://doi.org/10.1016/j.future.2013.01.010 Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications âĂŤ Big Data, Scalable Analytics, and Beyond.

[25] Josiah Hester and Jacob Sorber. 2017. Flicker: Rapid Prototyping for the Batteryless Internet-of-Things. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 19.

[26] Josiah Hester, Kevin Storer, and Jacob Sorber. 2017. Timely Execution on Intermittently Powered Batteryless Sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SenSys '17)*. ACM, New York, NY, USA, Article 17, 13 pages. https://doi.org/10.1145/3131672.3131673

[27] Jenalea Howell. 2017. https://technology.ihs.com/596542/number-of-connected-iot-devices-will-surge-to-125-billion-by-2030-ihs-markit-says. (2017).

[28] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava, and Vijay Raghunathan. 2006. Adaptive duty cycling for energy harvesting systems. In *Proceedings of the 2006 international symposium on Low power electronics and design*. ACM, 180–185.

[29] Roy Chaoming Hsu, Cheng-Ting Liu, and Wei-Ming Lee. 2009. Reinforcement learning-based dynamic power management for energy harvesting wireless sensor network. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer.

[30] R. C. Hsu, C. T. Liu, and H. L. Wang. 2014. A Reinforcement Learning-Based ToD Provisioning Dynamic Power Management for Sustainable Operation of Energy Harvesting Wireless Sensor Node. *IEEE Transactions on Emerging Topics in Computing* 2, 2 (June 2014), 181–191.

[31] R. C. Hsu, C. T. Liu, K. C. Wang, and W. M. Lee. 2009. QoS-Aware Power Management for Energy Harvesting Wireless Sensor Network Utilizing Reinforcement Learning. In *2009 International Conference on Computational Science and Engineering*, Vol. 2. 537–542.

[32] http://idealsciences.com/. 2018. https://www.amazon.com/Wireless-Temperature-Monitoring-Unlimited-Historical/dp/B01LXGFX5G. (2018).

[33] Hrishikesh Jayakumar, Kangwoo Lee, Woo Suk Lee, Arnab Raha, Younghyun Kim, and Vijay Raghunathan. 2014. Powering the internet of things. In *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 375–380.

[34] Shivaram Kalyanakrishnan and Peter Stone. 2007. Batch reinforcement learning in a complex domain. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 94.

[35] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B. Srivastava. 2007. Power Management in Energy Harvesting Sensor Networks. *ACM Trans. Embed. Comput. Syst.* 6, 4, Article 32 (Sept. 2007). https://doi.org/10.1145/1274858.1274870

[36] Bryce Kellogg, Aaron Parks, Shyamnath Gollakota, Joshua R Smith, and David Wetherall. 2014. Wi-Fi backscatter: Internet connectivity for RF-powered devices. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 607–618.

[37] Victor Lawson and Lakshmish Ramaswamy. 2015. Data Quality and Energy Management Tradeoffs in Sensor Service Clouds. In *Big Data (BigData Congress), 2015 IEEE International Congress on*. IEEE, 749–752.

[38] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent Computing: Challenges and Opportunities. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 71. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[39] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: Intermittent Execution Without Checkpoints. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 96 (Oct. 2017), 30 pages. https://doi.org/10.1145/3133920

[40] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 50–56.

[41] Paul Martin, Zainul Charbiwala, and Mani Srivastava. 2012. DoubleDip: Leveraging Thermoelectric Harvesting for Low Power Monitoring of Sporadic Water Use. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys '12)*. New York, NY, USA, 225–238.

[42] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. 2019. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT express* 5, 1 (2019), 1–7.

[43] MicroDAQ. 2019. www.microdaq.com/onset-hobo-bluetooth-humidity-data-logger.php. (2019).

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

[45] C. Moser, L. Thiele, D. Brunelli, and L. Benini. 2010. Adaptive Power Management for Environmentally Powered Systems. *IEEE Trans. Comput.* 59, 4 (April 2010), 478–491. https://doi.org/10.1109/TC.2009.158

[46] Saman Naderiparizi, Aaron N Parks, Zerina Kapetanovic, Benjamin Ransford, and Joshua R Smith. 2015. WISPCam: A battery-free RFID camera. In *RFID (RFID), 2015 IEEE International Conference on*. IEEE, 166–173.

[47] Saman Naderiparizi, Yi Zhao, James Youngquist, Alanson P. Sample, and Joshua R. Smith. 2015. Self-localizing Battery-free Cameras. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*. ACM, New York, NY, USA, 445–449.

[48] Panasonic. 2018. https://industrial.panasonic.com/cdbs/www-data/pdf/RDG0000/ABC0000C87.pdf. (2018).

[49] products.currentbyge.com. 2017. https://products.currentbyge.com/sites/products.currentbyge.com/files/documents/document_file/DT106-GE-Wireless-Occupancy-Sensor-Wall-Mount-Installation-Guide.pdf. (2017).

[50] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. 2008. A comparison of high-level full-system power models. *HotPower* 8, 2 (2008), 32–39.

[51] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. 1889–1897.

[52] Shaswot Shresthamali, Masaaki Kondo, and Hiroshi Nakamura. 2017. Adaptive Power Management in Solar Energy Harvesting Sensor Node Using Reinforcement Learning. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 5s (2017), 181.

[53] Sujesha Sudevalayam and Purushottam Kulkarni. 2011. Energy harvesting sensor nodes: Survey and implications. *IEEE Communications Surveys & Tutorials* 13, 3 (2011), 443–461.

[54] Richard S Sutton, Andrew G Barto, Francis Bach, et al. 1998. *Reinforcement learning: An introduction*. MIT press.

[55] Vamsi Talla, Bryce Kellogg, Shyamnath Gollakota, and Joshua R Smith. 2017. Battery-free cellphone. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 2 (2017), 25.

[56] Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10, Jul (2009), 1633–1685.

[57] Adrian Udenze and Klaus McDonald-Maier. 2009. Direct reinforcement learning for autonomous power configuration and control in wireless networks. In *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*. IEEE, 289–296.

[58] Sennur Ulukus, Aylin Yener, Elza Erkip, Osvaldo Simeone, Michele Zorzi, Pulkit Grover, and Kaibin Huang. 2015. Energy harvesting wireless communications: A review of recent advances. *IEEE Journal on Selected Areas in Communications* 33, 3 (2015), 360–381.

[59] utc.com. 2018. https://www.utc.com/. (2018).

[60] Christopher M Vigorito, Deepak Ganesan, and Andrew G Barto. 2007. Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. IEEE, 21–30.

[61] Qin Wang, Mark Hempstead, and Woodward Yang. 2006. A realistic power consumption model for wireless sensor network devices. In *2006 3rd annual IEEE communications society on sensor and ad hoc communications and networks*, Vol. 1. IEEE, 286–295.

[62] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[63] Christopher John Cornish Hellaby Watkins. 1989. *Learning from delayed rewards*. Ph.D. Dissertation. King's College, Cambridge.

[64] Zhenchun Wei, Fei Liu, Zengwei Lyu, Xu Ding, Lei Shi, and Chengkai Xia. 2018. Reinforcement learning for a novel mobile charging strategy in wireless rechargeable sensor networks. In *International Conference on Wireless Algorithms, Systems, and Applications*. Springer, 485–496.

[65] www.enocean.com. 2019. Enocean Motion Detector. https://www.enocean.com/en/products/enocean_modules_24ghz_ble/easyfit-motion-detector-with-illumination-sensor-emdcb/. (2019).

[66] David K. Y. Yau, Nung Kwan Yip, Chris Y. T. Ma, Nageswara S. V. Rao, and Mallikarjun Shankar. 2010. Quality of Monitoring of Stochastic Events by Periodic and Proportional-share Scheduling of Sensor Coverage. *ACM Trans. Sen. Netw.* 7, 2, Article 18 (Sept. 2010), 49 pages.

[67] Kok-Lim Alvin Yau, Peter Komisarczuk, and Paul D Teal. 2012. Reinforcement learning for context awareness and intelligence in wireless networks: Review, new features and open issues. *Journal of Network and Computer Applications* 35, 1 (2012), 253–267.