Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 765182, 11 pages
http://dx.doi.org/10.1155/2014/765182

Research Article

Performance Analysis of Resource-Aware Task Scheduling Methods in Wireless Sensor Networks

Muhidul Islam Khan and Bernhard Rinner

Institute of Networked and Embedded Systems, Alpen-Adria-University of Klagenfurt, 9020 Klagenfurt, Austria

Correspondence should be addressed to Muhidul Islam Khan; muhidulislam.khan@aau.at

Received 24 June 2014; Revised 29 August 2014; Accepted 1 September 2014; Published 22 September 2014

Academic Editor: King-Shan Lui

Wireless sensor networks (WSNs) are an attractive platform for monitoring and measuring physical phenomena. WSNs typically consist of hundreds or thousands of battery-operated tiny sensor nodes which are connected via a low data rate wireless network. A WSN application, such as object tracking or environmental monitoring, is composed of individual tasks which must be scheduled on each node. Naturally the order of task execution influences the performance of the WSN application. Scheduling the tasks such that the performance is increased while the energy consumption remains low is a key challenge. In this paper we apply online learning to task scheduling in order to explore the tradeoff between performance and energy consumption. This helps to dynamically identify effective scheduling policies for the sensor nodes. The energy consumption for computation and communication is represented by a parameter for each application task. We compare resource-aware task scheduling based on three online learning methods: independent reinforcement learning (RL), cooperative reinforcement learning (CRL), and exponential weight for exploration and exploitation (Exp3). Our evaluation is based on the performance and energy consumption of a prototypical target tracking application. We further determine the communication overhead and computational effort of these methods.

1. Introduction

A wireless sensor network (WSN) is an attractive platform for various applications including target tracking, environmental monitoring, data aggregation, and smart environments. The application is composed of tasks which need to be executed during the operation on the sensor nodes. The sensor nodes are typically supplied by batteries and thus pose strong limitations not only on energy but also on computation, storage, and communication capabilities [1–4].

The scheduling of the individual tasks has a strong influence on the achievable performance and energy consumption. WSNs operate in a dynamic environment where the need for adaptive and autonomous task scheduling is well recognized [5]. Since it is not possible to schedule the tasks a priori, online, and resource-aware task scheduling is required for a WSN. For determining the next task to execute, the sensor nodes need to consider the impact of each available task on the energy budget and the application's performance. There is tradeoff between application performance and resource consumption, and the task scheduler of the node should be able to adapt to changes in the environment. For example, in a target tracking application, sensor nodes should frequently execute the tracking task when objects are within the field of view (FOV). Since tracking is very resource-consuming, this task should be avoided when no object to track is nearby. Thus, task scheduling is an important issue to improve the energy/performance tradeoff, and we investigate scheduling methods which are able to learn effective scheduling strategies in dynamic environments. We also investigate the effect of cooperation/communication among neighboring nodes with the local observations for task scheduling which is typical in a WSN. Cooperation among neighboring nodes has an impact on the overall application state and is able to further improvement on the energy/performance tradeoff. Since resource-awareness is an important aspect, we consider energy consumption for the tasks scheduling and aim for low resource consumption of the scheduling algorithms.

In this paper, we apply online learning to task scheduling in order to explore the tradeoff between performance and energy consumption. We compare resource-aware task

scheduling based on three online learning methods: independent reinforcement learning (RL), cooperative reinforcement learning (CRL), and exponential weight for exploration and exploitation (Exp3). Our evaluation is based on a simulation study of the performance and energy consumption of a prototypical target tracking application. We further determine the communication overhead and computational effort of these methods.

The rest of this paper is organized as follows. Section 2 discusses related work, and Section 3 introduces the problem formulation. Section 4 describes the underlying system model for task scheduling based on online learning. In Section 5 we present the three task scheduling methods. Section 6 presents the experimental setup and discusses the simulation results for a target tracking application. Section 7 concludes this paper with a summary and brief discussion on future work.

2. Related Works

In a resource-constrained WSN, effective task scheduling is very important for facilitating the effective usage of resources [6]. The cooperative behavior among sensor nodes by exchanging data among neighboring nodes can be very helpful to schedule the tasks in a way that the energy consumption is reduced and also a considerable performance is maintained. Most of the existing methods of tasks scheduling in WSN do not provide online scheduling of tasks. They mainly consider static task allocation instead of focusing on distributed task scheduling. The main difference between task allocation and task scheduling is that task allocation deals with the problem of determining a set of task assignments on a sensor network that minimizes an objective function such as the total execution time [7, 8]. On the other hand, the objective of task scheduling is to determine the best temporal order of tasks for each sensor node. In offline scheduling, the complete information about the system activities is available a priori, and the schedule can be determined at compile time. Due to the high dynamics of WSN complete system information is only available at runtime which requires online scheduling [9]. In the following paragraphs we discuss related task scheduling approaches for WSN and stress the key differences in the presented approach.

Guo et al. [10] propose a self-adaptive task allocation strategy in a WSN. They assume that the WSN is composed of a number of sensor nodes and a set of independent tasks which compete for the sensors. They neither consider distributed tasks scheduling nor the tradeoff among energy consumption and performance.

Giannecchini et al. [11] propose an online task scheduling mechanism called collaborative resource allocation to allocate the network resources between the tasks of periodic applications in WSNs. This mechanism also does not explicitly consider energy consumption.

Frank and Römer [6] propose an algorithm for generic task allocation in wireless sensor networks. They define some rules for the task execution and propose a role-rule model for sensor networks where "role" is used as a synonym for task. It is a programming abstraction of the role-rule model. This distributed approach provides a specification that defines possible roles and rules for how to assign roles to nodes. This specification is distributed to the whole network via a gateway or alternatively it can be preinstalled on the nodes. A role assignment algorithm takes into account the rules and node properties, which may trigger execution and in network data aggregation. This generic role assignment approach does consider the energy consumption but not the ordering of tasks to sensor nodes.

Krishnamachari et al. [12] examine the channel utilization as resource management problem by a distributed constraint satisfaction method. They consider a wireless sensor network of n nodes placed randomly in a square area with a uniform, independent distribution. This work tests three self-configuration tasks in wireless sensor networks: partition into coordinating cliques, formation of Hamiltonian cycles, and conflict-free channel scheduling. They explore the impact of varying the transmission radius on the solvability and complexity of these problems. In the case of partition into cliques and Hamiltonian cycle formation, they observe that the probability that these tasks can be performed undergoes a transition from zero to one. This constraint satisfaction approach neither addresses mapping of tasks to sensor nodes nor discusses the resource consumption/performance tradeoff.

Dhanani et al. [13] compare utility-based information management policies in sensor networks. Here, the considered resource is information or data, and two models are distinguished: the sensor-centric utility-based model (SCUB) and the resource manager (RM) model. SCUB follows a distributed approach that instructs individual sensors to make their own decisions about what sensor information should be reported based on an utility model for data. RM is a consolidated approach that takes into account knowledge from all sensors before making decisions. They evaluate these policies through simulation in the context of dynamically deployed sensor networks in military scenarios. Both SCUB and RM can extend the lifetime of a network as compared to a network without maintaining any policy. This approach does not address the task scheduling to improve the resource consumption/performance tradeoff.

Shah and Kumar [14] introduce a task scheduling approach for WSN based on an independent reinforcement learning algorithm (RL) for online tasks scheduling. They use Q learning [15] for the task scheduling. Their approach relies on a simple and fixed network topology consisting of three nodes and a static value for the reward function. They further consider neither any cooperation among neighbors nor the energy/performance tradeoff.

In our previous work [16] we applied cooperative reinforcement learning (CRL) for online tasks scheduling. We used SARSA(λ) [17] learning and introduced cooperation among neighboring sensor nodes to further improve the task scheduling. In this paper, we introduce exponential bandit solvers to online task scheduling; that is, we apply Exp3 (exponential weight for exploration and exploitation) [18] which is an adversarial or nonstochastic bandit solver. We compare RL, CRL, and Exp3 for the tasks scheduling in a

target tracking application and analyze the performance in terms of tracking quality/energy consumption tradeoff. The proposed approach also considers the cooperation where each node shares local observations of object trajectories with the neighboring nodes.

3. Problem Formulation

In our approach the WSN is composed of N nodes represented by the set $\widehat{N} = \{n_1, \ldots, n_N\}$. We abstract the deployment of the WSN by a simple 2D space where each node has a known position (u_i, v_i) and a given sensing coverage range which is simply represented by a circle with radius r_i. All nodes within the communication range R_i can directly communicate with n_i and are referred to as neighbors. The number of neighbors of n_i is given as $ngh(n_i)$. The available energy of node n_i is modeled by a scalar value E_i.

The WSN application is composed of A independent tasks (or actions) represented by the set $\widehat{A} = \{a_1, \ldots, a_A\}$. Once a task is started at a specific node, it is executed for a specific (short) period of time and terminates afterwards. Each task execution on a specific node n_i requires some energy $\widetilde{E}_j$ either for computation or communication and contributes to the overall application performance P. The energy consumption for computation and communication is represented by $\widetilde{E}_j$ for processing tasks and communication tasks, respectively. Thus, the execution of task a_j on node n_i is only feasible if $E_i \geq \widetilde{E}_j$. The set $\widehat{A}$ is known at the initiation of the applications and does not change during the operation. The set of available nodes $\widehat{N}$ can change during operation, for example, due to completion of its energy source. The overall performance P is represented by an application-specific metric (cp. Section 4 for more details). On each node, an online task scheduling takes place which selects the next task to execute among the A independent tasks. The task execution time is abstracted as a fixed period. Thus, scheduling is required at the end of each period which is represented as time instant t_i. We only consider nonpreemptive scheduling.

The ultimate objective for our problem is to determine the order of tasks on each node such that the overall performance is maximized while the energy consumption is minimized.

4. System Model

The task scheduler operates in a highly dynamic environment, and the effect of the task ordering on the overall application performance is difficult to model. Figure 1 depicts our scheduling framework where its key components can be described as follows.

(i) *Agent*: each sensor node embeds an agent which is responsible for executing the online learning algorithm.

(ii) *Environment*: the WSN application represents the environment in our approach. Interaction between the agent and the environment is achieved by executing actions and receiving a reward function.

(iii) *Action*: an agent's action is the currently executed application task on the sensor node. At the end of each time period t_i each node triggers the scheduler to determine the next action to execute.

(iv) *State*: a state describes an internal abstraction of the application which is typically specified by some system parameters. In our target tracking application, the states are represented by the number of currently detected targets in the node's FOV and expected arrival times of targets detected by neighboring nodes. The state transitions depend on the current state and action.

(v) *Policy*: an agent's policy determines what action will be selected in a particular state. In our case, this policy determines which task to execute at the perceived state. The policy can focus more on exploration or exploitation depending on the selected setting of the learning algorithm.

(vi) *Value function*: this function defines what is good for an agent over the long run. It is built upon the reward function values over time and hence its quality totally depends on the reward function [14].

(vii) *Reward function*: this function provides a mapping of the agent's state and the corresponding action to a reward value that contributes to the performance. We apply a weighted reward function which is capable of expressing the tradeoff between energy consumption and tracking performance.

(viii) *Cooperation*: we consider the information exchange among neighboring nodes as cooperation. The received information may influence the application's state of a sensor nodes.

4.1. Set of Actions. We consider the following actions in our target tracking application.

(a) *Detect_Targets*: this function scans the field of view (FOV) and returns the number of detected targets in the FOV.

(b) *Track_Targets*: this function keeps track of the targets inside the FOV and returns the current 2D positions of all targets. Every target within the FOV is assigned with a unique ID number.

(c) *Send_Message*: this function sends information about the target's trajectory to neighboring nodes. The trajectory information includes (i) the current position and time of the target and (ii) the estimated speed and direction. This function is executed when the target is about to leave the FOV.

(d) *Predict_Trajectory*: this function predicts the velocity of the trajectory. A simple approach is to use the two most recent target positions, that is, (x_t, y_t) at time

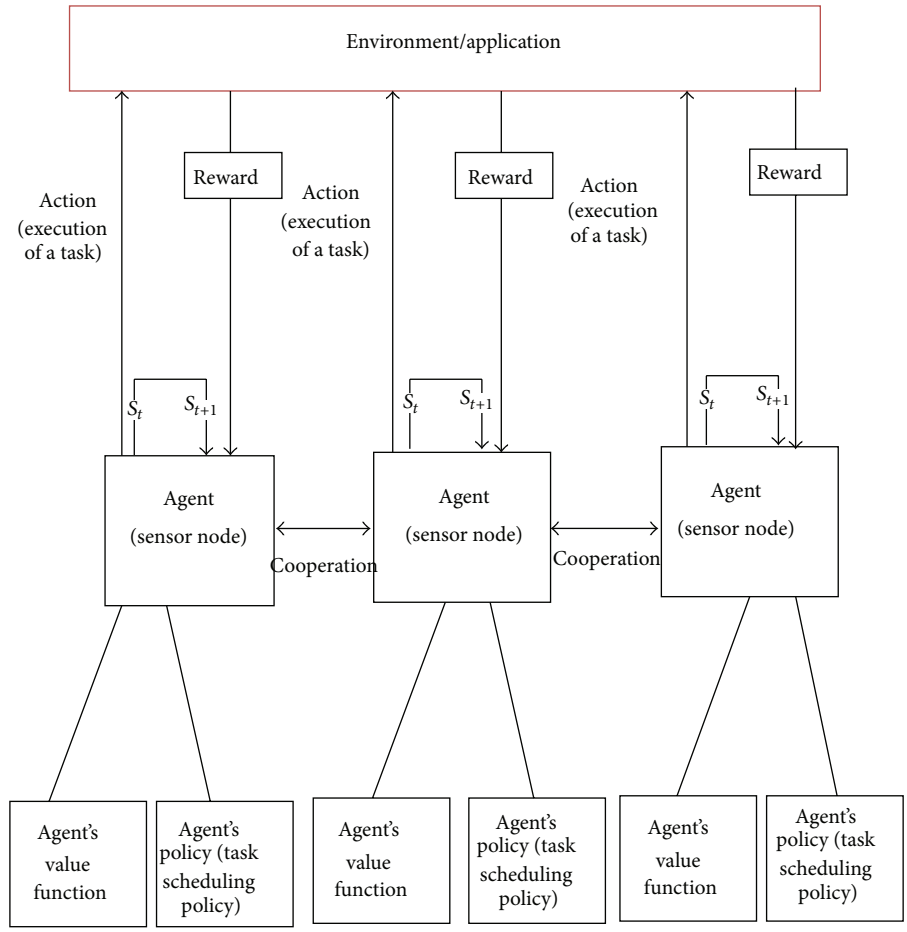

Figure 1: General framework for task scheduling using online learning.

t_t and (x_{t-1}, y_{t-1}) at t_{t-1}. Then the constant target's speed can be estimated as

$$v = \frac{\sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2}}{t_t - t_{t-1}}. \quad (1)$$

(e) *Intersect_Trajectory*: this function checks whether the trajectory intersects with the FOV and predicts the expected time of the intersection. This function is executed by all nodes which receive the "target trajectory" information from a neighboring node. Trajectory intersection with the FOV of a sensor node is computed by basic algebra. The expected time to intersect the node is estimated by

$$\widetilde{t}_i = \frac{D_{P_i P_j}}{v}, \quad (2)$$

where $D_{P_i P_j}$ is the distance between points P_j and P_i. P_j represents the point where the trajectory is predicted at node j and P_i corresponds to the trajectory's intersection points with the FOV of node i (cp. Figure 2). v is the estimated velocity as calculated by (1).

(f) *Goto_Sleep*: this function shuts down the sensor node for single time period. It consumes the least amount of energy of all available actions.

4.2. Set of States. We abstract the application by the following three states at every node.

(i) *Idle*: this state indicates that there is currently no target detected within the node's FOV and the local clock is too far from the expected arrival time of any target already detected by some neighbor. If the time gap between the local clock L_c and the expected arrival time N_{ET} is greater than or equal to a threshold Th_1 (cp. Figure 3), then the node remains in the idle state. The threshold Th_1 is set to 5 based on our simulation studies. In this state, the sensor node performs *Detect_Targets* less frequently to save energy.

(ii) *Awareness*: there is currently also no detected target in the node's FOV in this state. However, the node has received some relevant trajectory information and the expected arrival time of at least one target is in less than Th_1 clock ticks. In this state, the sensor node performs *Detect_Targets* more frequently, since at least one target is expected to enter the FOV.

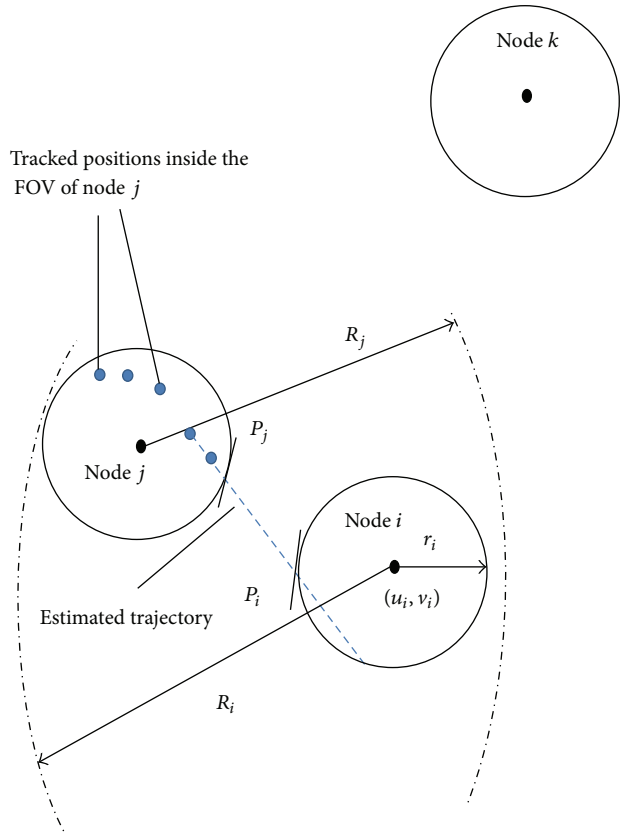

Figure 2: Target prediction and intersection. Node j estimates the target trajectory and sends the trajectory information to its neighbors. Node i checks whether the predicted trajectory intersects with its FOV and computes the expected arrival time.

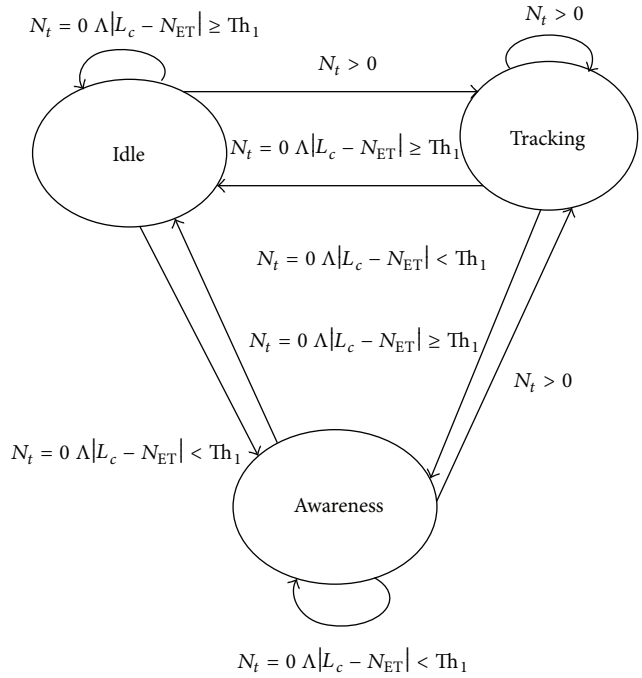

Figure 3: State transition diagram. States change according to the value of two application variables N_t and N_{ET}. L_c represents the local clock value and Th_1 is a time threshold.

(iii) *Tracking*: this state indicates that there is currently at least one detected target within the node's FOV. Thus, the sensor node performs tracking frequently to achieve high tracking performance.

Obviously, the frequency of executing *Detect_Targets* and *Track_Targets* depends on the overall objective, that is, whether to focus more on tracking performance or energy consumption. The states can be identified by two application variables, that is, the number of detected targets at the current time N_t and the list of arrival times of targets expected to intersect with node N_{ET}. N_t is determined by the task *Detect_Targets* which is executed at time t. If the sensor node executes the task *Detect_Targets* at time t then N_t returns the number of detected targets in the FOV. Each node maintains a list of appearing targets and the corresponding arrival time. Targets are inserted in this list if the sensor node receives a message and the estimated trajectory intersects with the FOV. Targets are removed if a target is detected by the node or the expected arrival time with an additional threshold Th_1 has expired.

Initially each node has no idea about which task to perform at which state. They learn this scheduling online over time. For example, *Track_Targets* is necessary task to keep tracking when the target is in FOV. The application learns online about the next task to execute based on our proposed methods. If the sensor node does not perform the *Track_Targets* task when the target is in FOV, there is a chance to miss the target which implies less tracking quality. But this situation could provide better energy efficiency, since *Track_Targets* task consumes the highest amount of energy among all the tasks. So, selection of a particular task at each time step or scheduling of tasks provides an impact on overall tracking quality/energy consumption tradeoff.

Figure 3 depicts the state transition diagram where L_c is the local clock value of the sensor node and Th_1 represents the time threshold between L_c and N_{ET}.

4.3. Reward Function. The reward function is a key system component for expressing the effect of the task execution on the system performance and resource consumption. Thus, both aspects should be covered by the reward function. Among the various options we simply merge energy consumption and system performance using a balancing parameter. In detail, the reward function in our algorithm is defined as

$$r = \beta\left(\frac{E_i}{E_{\max}}\right) + (1-\beta)\left(\frac{P_t}{P}\right), \quad (3)$$

where the parameter β balances the conflicting objectives between E_i and P_t. E_i represents the residual energy of the node. P_t represents the number of tracked positions of the target inside the FOV of the node. $E_{\max}$ is the maximum energy level of sensor node and P is the number of all possible detected target's positions in the FOV. These two parameters

(1) Initialize $Q(s, a) = 0$. Where s is the set of states and a is the set of actions
(2) **while** Residual energy is larger than zero **do**
(3) Determine current state s by application variables
(4) Select an action a which has the highest Q value
(5) Execute the selected action
(6) Calculate Q value for the executed action
(7) Calculate the value function for the executed action
(8) Shift to next state based on the executed action
(9) **end while**

Algorithm 1: Q learning for task scheduling.

are used for normalizing the energy and performance parameters. By modifying the balancing parameter β we can control whether more focus is put on energy efficiency or system performance.

5. Online Learning Methods for Task Scheduling

We use RL, CRL, and Exp3 for the task scheduling in a multitarget tracking application. These three methods are online machine learning methods for task scheduling. RL and CRL are reinforcement learning methods. In RL, we do not exchange information among neighboring nodes. In CRL and Exp3, we exploit cooperation by exchanging trajectory information among neighboring nodes. In the following subsections we briefly describe the three learning methods and explain their key parameters. For our experiments we abstract the tracking application with the same tasks, states, and reward function.

5.1. Independent Reinforcement Learning (RL). RL task scheduling follows the work of Shah and Kumar [14] which uses traditional Q learning [15] as online learning strategy. In Q learning the scheduling policy is represented by a two-dimensional matrix $Q_{t+1}(s, a)$ indexed by state-action pairs. The optimal Q value for a particular action in a particular state is the sum of the reinforcement received when that action is taken and the discounted best Q value for the state that is reached by taking that action [15].

The main idea of RL is to allow each individual sensor node to self-schedule its tasks and allocate its resources by learning their usefulness in any given state while honoring the application defined constraints and maximizing the total amount of reward over time.

In Q learning every agent needs to maintain a Q matrix for the value functions. Initially all entries of the Q matrix are zero and the agent of the nodes may be in any state. Based on the application defined variables, the system goes to a particular state. Then it performs an action which depends on the status of the nodes. It calculates the Q value for this (state, action) pair as

$$Q_{t+1}(s_t, a_t) = (1 - \alpha) Q_t(s_t, a_t) + \alpha (r_{t+1}(s_{t+1}) + \gamma V_t(s_{t+1})),$$
$$V_{t+1}(s_t) = \max_{a \in A} Q_{t+1}(s_t, a), \tag{4}$$

where $Q_{t+1}(s_t, a_t)$ means the update of the Q value at time $t + 1$ after executing the action a at time step t. r_{t+1} represents the immediate reward after executing the action a at time t, V_t represents the value function for node at time t, and V_{t+1} represents the value function at time $t + 1$. $\max_{a \in A} Q_{t+1}(s_t, a)$ means the maximum Q value after performing an action from the action set A for the agent i. γ is the discount-factor which can be set to a value in $[0, 1]$. For higher γ values, the agent relies more on the future than the immediate reward. α is the learning rate parameter which can be set to a value in $[0, 1]$. It controls the rate at which an agent tries to learn by giving more or less weight to the previously learned utility value. When α is close to 1, the agent gives more priority to the previously learned utility value.

Algorithm 1 depicts the RL algorithm.

5.2. Cooperative Reinforcement Learning (CRL). The CRL task scheduling follows a cooperative SARSA(λ) learning algorithm. SARSA(λ) [17], also referred to as State-Action-Reward-State-Action, is an iterative algorithm that approximates the optimal solution without knowledge of the transition probabilities which is very important for a dynamic system like WSN. At each state s_{t+1} of iteration $t + 1$, it updates $Q_{t+1}(s, a)$, which is an estimate of the Q function by computing the estimation error δ_t after receiving the reward in the previous iteration. The SARSA(λ) algorithm has the following update rule for the Q values:

$$Q_{t+1}(s_t, a_t) \longleftarrow Q_t(s, a) + \alpha \delta_t e_t(s_t, a_t), \tag{5}$$

for all s, a.

In (5), $\alpha \in [0, 1]$ is the learning rate which decreases with time. δ_t is the temporal difference error which is calculated by following rule:

$$\delta_t = r_{t+1} + \gamma_1 f Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t). \tag{6}$$

In (6), γ_1 is a discount-factor which varies from 0 to 1. The higher the value is, the more the agent relies on future rewards than on the immediate reward. r_{t+1} represents the reward received for performing action. f is the weight factor [19] for the neighbors of agent i and can be defined as follows:

$$f = \frac{1}{ngh(n_i)} \quad \text{if } ngh(n_i) \neq 0,$$
$$f = 1 \quad \text{otherwise.} \tag{7}$$

An important aspect of an RL-framework is the tradeoff between exploration and exploitation [20]. Exploration deals with randomly selecting actions which may not have higher utility in search of better rewarding actions, while exploitation aims at the learned utility to maximize the agent's reward.

In our proposed algorithm, we use a simple heuristic where exploration probability at any point of time is given by

$$\epsilon = \min\left(\epsilon_{\max}, \epsilon_{\min} + k * \frac{S_{\max} - S}{S_{\max}}\right), \tag{8}$$

where $\epsilon_{\max}$ and $\epsilon_{\min}$ define upper and lower boundaries for the exploration factor, respectively. $S_{\max}$ represents the maximum number of states which is three in our work and S represents the current number of states already known. At each time step, the agent calculates ϵ and generates a random number in the interval of $[0, 1]$. If the selected random number is less than or equal to ϵ, the system chooses a uniformly random task (exploration); otherwise it chooses the best task using Q values (exploitation).

SARSA(λ) improves learning through eligibility traces $e_t(s, a)$ (cp. (5)). Here λ is another learning parameter similar to α for guaranteed convergence. γ_2 is the discount-factor. In general, eligibility traces give a higher update factor for recently revisited states. This means that the eligibility trace for a state-action pair (s, a) will be reinforced if $s_t \in s$ and $a_t \in a$. Otherwise, if the previous action a_t is not greedy, the eligibility trace is cleared.

The eligibility trace is updated by the following rule:

$$e_t(s_t, a_t) = \gamma_2 \lambda e_{t-1}(s_t, a_t) + 1 \quad \text{if } s_t \in s,\ a_t \in a, \tag{9}$$

$$e_t(s_t, a_t) = \gamma_2 \lambda e_{t-1}(s_t, a_t) \quad \text{otherwise.} \tag{10}$$

Algorithm 2 depicts the cooperative SARSA(λ) learning algorithm.

The learning rate α is decreased slowly in such a way that it reflects the degree at which a state-action pair has been chosen in the recent past. It is calculated as

$$\alpha = \frac{\zeta}{\text{visited}(s, a)}, \tag{11}$$

where ζ is a positive constant. visited (s, a) represents the visited state-action pairs so far [21].

5.3. Bandit Solvers (Exp3). We use the classical adversarial algorithm Exp3 (Exponential-weight algorithm for Exploration and Exploitation) for the task scheduling [18].

Algorithm 3 depicts the bandit solver algorithm Exp3.

In Exp3 the parameter κ controls the probability with which arms are explored in each round. At each time step t, Exp3 draws an action a according to the distribution $P_{1,t}, P_{2,t}, \ldots, P_{A,t}$. This distribution is a mixture of the uniform distribution and a distribution which assigns to each action a probability mass exponential in the estimated reward for that action. Intuitively, mixing the uniform distribution is done to make sure that the algorithm tries out all A actions and gets good estimates of the rewards for each action.

(1) Initialize $Q(s, a) = 0$ and $e(s, a) = 0$
(2) **while** Residual energy is larger than zero **do**
(3) Determine current state s by application variables
(4) Select an action a, using policy
(5) Execute the selected action
(6) Calculate reward for the executed action (3)
(7) Update the learning rate (11)
(8) Calculate the temporal difference error (6)
(9) Update the eligibility traces (10)
(10) Update the Q value (5)
(11) Shift to next state based on the executed action
(12) **end while**

Algorithm 2: SARSA(λ) learning algorithm for target tracking application.

Exp3 works by maintaining a list of weights w_i for each of the actions, by using these weights to decide which action to take next based on a probability distribution P_t and by increasing the relevant weights when the reward is positive. The egalitarianism factor $\kappa \in [0, 1]$ tunes the desire to pick an action uniformly at random. If $\kappa = 1$, the weights have no effect on the choices at any step.

6. Experimental Results and Evaluation

We evaluate the task scheduling methods using a WSN multitarget tracking scenario implemented in a C# simulation environment. The simulator consists of two stages: the deployment of the nodes and the execution of the tracking application. In our evaluation scenario the sensor nodes are uniformly distributed in a 2D rectangular area. A given number of sensor nodes are placed randomly on this area which can result in partially overlapping FOVs of the nodes. However, placement of nodes on the same position is avoided. Network parameters such as the number of nodes, the sensor radius, and the transmission radius can be configured in our simulator. Once these network parameters are configured, we can run the simulation with our selected algorithm.

Targets move around in the area based on a Gauss-Markov mobility model [22]. The Gauss-Markov mobility model was designed to adapt to different levels of randomness via tuning parameters. Initially, each mobile target is assigned a current speed and direction. At each time step t, the movement parameters of each target are updated based on the following rule:

$$\begin{aligned} S_t &= \eta S_{t-1} + (1 - \eta) S + \sqrt{1 - \eta^2} S_{t-1}^G, \\ D_t &= \eta D_{t-1} + (1 - \eta) D + \sqrt{1 - \eta^2} D_{t-1}^G, \end{aligned} \tag{12}$$

where S_t and D_t are the current speed and direction of the target at time t. S and D are constants representing the mean value of speed and direction. S_{t-1}^G and D_{t-1}^G are random variables from a Gaussian distribution. η is a parameter in the range $[0, 1]$ and is used to vary the randomness of the motion. Random (Brownian) motion is obtained if $\eta = 0$,

(1) Parameters: Number of tasks A, Factor $\kappa \leq 1$
(2) Initialization: $w_{i,0} = 1$ and $P_{i,1} = 1/A$ for $i = 1, 2, \ldots, A$
(3) **while** Residual energy is larger than zero **do**
(4) Determine current s based on application variables
(5) Select an action $a \in \{1, 2, \ldots, A\}$ based on the P_t
(6) Execute the selected action
(7) Calculate the reward: $r_{t+1}/P_{a,t}$
(8) Update the weights: $w_{a,t} = w_{a,t-1}e^{\kappa r_{t+1}}$
(9) Calculate the updated probability distribution:
(10) $$P_{j,t+1} = (1-\kappa)\frac{w_{a,t}}{\sum_{j=1}^{A} w_{j,t}} + \frac{\kappa}{A}, \quad j = 1, 2, \ldots, A$$
(11) Shift to next state based on the executed action
(12) **end while**

Algorithm 3: Task scheduling by bandit solver Exp3.

Table 1: Energy consumption of the individual actions.

Action	Energy consumption
Goto_Sleep	1 unit
Detect_Targets	2 units
Intersect_Trajectory	3 units
Predict_Trajectory	4 units
Send_Message	5 units
Track_Targets	7 units

and linear motion is obtained if $\eta = 1$. At each time t, the target's position is given by the following equations:

$$x_t = x_{t-1} + S_{t-1}\cos\left(D_{t-1}\right), \qquad y_t = y_{t-1} + S_{t-1}\sin\left(D_{t-1}\right). \tag{13}$$

In our simulation we limit the number of concurrently available targets to seven. The total energy budget for each sensor node is considered as 1000 units. Table 1 shows the energy consumption for the execution of each action. We set the discount-factors $\gamma = 0.5$, $\gamma_1 = 0.5$ and $\gamma_2 = 0.5$ for the online learning algorithms and vary the learning rate according to (11). We set $\zeta = 1$ for calculating learning rate in (11). We set $k = 0.25$, $\epsilon_{\min} = 0.1$, $\epsilon_{\max} = 0.3$, and $S_{\max} = 3$ in (8). We set $\lambda = 0.5$ for the eligibility trace calculation by (10). We set the egalitarianism factor $\kappa = 0.5$ for Exp3. We consider the sensing radius as $r_i = 5$ and communication radius as $R_i = 8$. We set these fixed values for the parameters based on our simulation studies. For each simulation run we aggregate the achieved tracking quality and energy consumption and normalize the tracking quality and energy consumption to $[0, 1]$.

For our evaluation we perform the following four experiments with the following assumptions of parameters.

(1) To find out the tradeoff between tracking quality and energy consumption, we set the balancing factor β of the reward function between $[0.1, 0.9]$ in 0.1 steps, keep the randomness of moving target as $\eta = 0.5$, set the egalitarianism factor of Exp3 as $\kappa = 0.5$, and fix the topology to five nodes.

(2) We vary the network size to check the tradeoff between tracking quality and energy consumption. We consider three different topologies consisting of 5, 10, and 20 sensor nodes where the coverage ratio is 0.0029, 0.0057, and 0.0113, respectively. The coverage ratio is defined as the ratio of the aggregated FOV of all deployed sensor nodes over the area of the entire surveillance area. We keep the balancing factor $\beta = 0.5$ and the randomness of the mobility model $\eta = 0.5$ constant for this experiment.

(3) We set the randomness of moving targets η to one of the following values $\{0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 0.7, 0.9\}$ and set the balancing factor $\beta = 0.5$ and fix the topology to five nodes.

(4) We evaluate RL, CRL, and Exp3 in terms of average execution time and average communication effort. These values are measured from twenty iterations and represent the mean execution times and the mean of *Send_Message* task executions.

Figure 4 shows the results of our first experiment. Each data point in these figures represents the average of normalized tracking quality and energy consumption of ten complete simulation runs. The results show the tracking quality/energy consumption tradeoff for RL, CRL, and Exp3 by varying the balancing factor β between $[0.1, 0.9]$ in 0.1 steps. We observe that CRL and Exp3 provide similar results; that is, the corresponding data points are closely colocated. RL is more energy-aware but is not able to achieve high tracking quality.

Figure 5 shows the results of our second experiment. In this experiment, each data point represents the average of normalized tracking quality and energy consumption of ten complete simulation runs by varying the network size to one of the values $\{5, 10, 20\}$ for each method. Here the same trend can be identified; that is, the CRL and Exp3 achieve almost

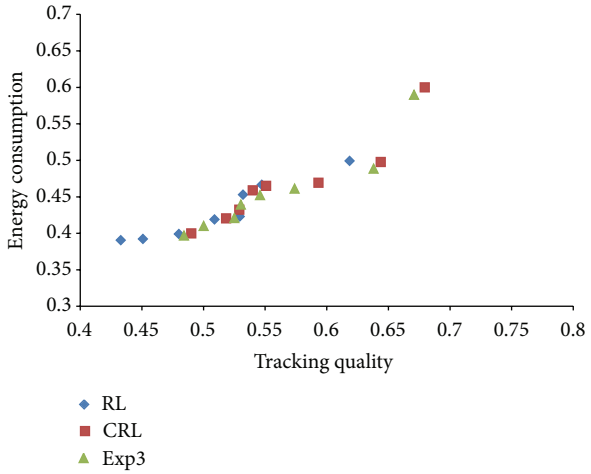

Figure 4: Tracking quality/energy consumption tradeoff for RL, CRL, and Exp3 by varying the balancing factor of the reward function β.

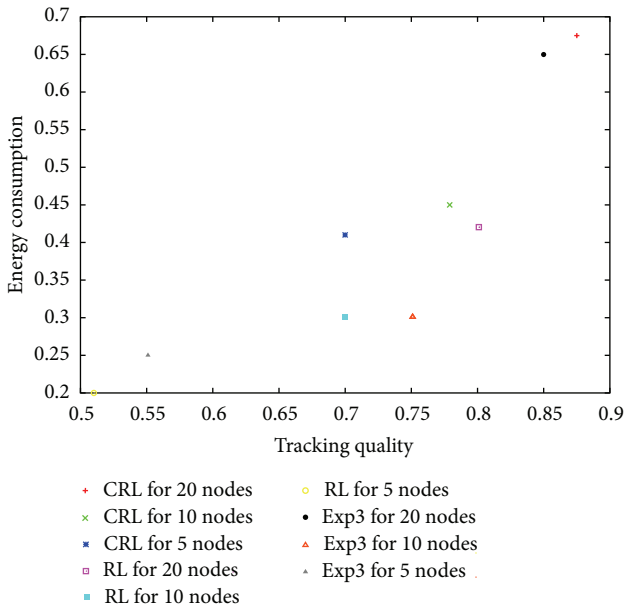

Figure 5: Tracking quality/energy consumption tradeoff for RL, CRL, and Exp3 by varying the network size.

similar results in terms of tracking quality/energy consumption tradeoff and RL shows less tracking performance with the higher energy efficiency.

Figures 6, 7, and 8 show the results of our third experiment. In this experiment, each data point represents the average of normalized tracking quality and energy consumption of ten complete simulation runs by varying the randomness of moving objects η to one of these values {0.10, 0.15, 0.20, 0.25, 0.30, 0.40, 0.50, 0.70, 0.90} for each method. From these figures, it can be seen that CRL and Exp3 outperform RL in terms of achieved tracking performance. We can see that, for lower randomness $\eta = 0.5$, 0.7, and 0.9, RL and Exp3 show very close results for tracking performance.

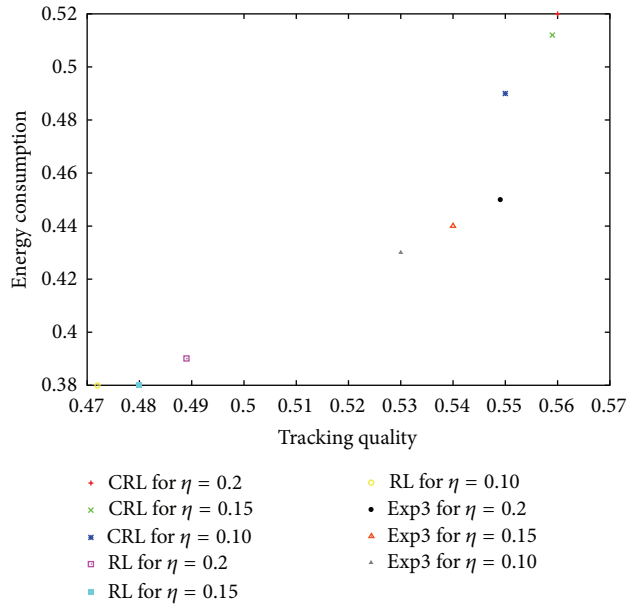

Figure 6: Tracking quality/energy consumption tradeoff for RL, CRL, and Exp3 by varying the randomness of target movement, $\eta = 0.10$, 0.15, and 0.20.

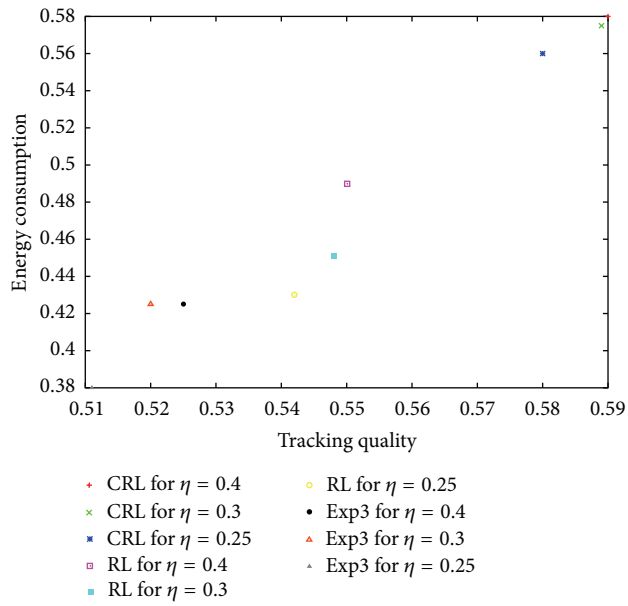

Figure 7: Tracking quality/energy consumption trade-off for RL, CRL and Exp3 by varying the randomness of target movement, $\eta = 0.25$, 0.30 and 0.40.

But, for higher randomness $\eta = 0.1$, 0.15, and 0.2, RL gives poor performance with regard to tracking performance.

Table 2 shows the comparison of RL, CRL, and Exp3 in terms of average execution time and average communication effort. These values are derived from twenty iterations and represent the mean execution times and the mean of *Send_message* task executions. We find that RL is the most resource-aware scheduling algorithm. Exp3 requires 25% and

Table 2: Comparison of average execution time and average number of transferred messages (based on 20 iterations).

	Avgerage execution time	Avgerage communication effort
RL	0.036 s	0
CRL	0.067 s	29
Exp3	0.045 s	27

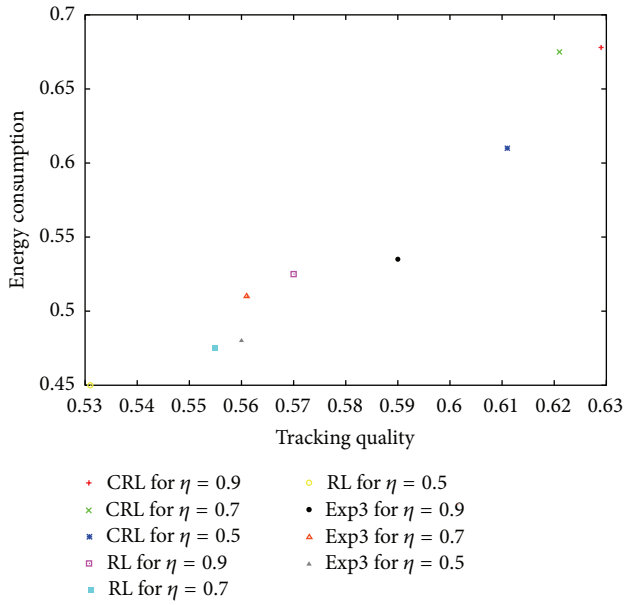

Figure 8: Tracking quality/energy consumption tradeoff for RL, CRL, and Exp3 by varying the randomness of target movement, $\eta = 0.50, 0.70$, and 0.90.

CRL requires 86% more execution time, respectively. The communication overhead is similar for both Exp3 and CRL.

7. Conclusion

In this paper we applied online learning algorithms for resource-aware task scheduling in WSNs. We analyzed and compared the performance of online task scheduling methods based on the three learning algorithms: RL, CRL, and Exp3. Our evaluation results show that these methods provide different properties concerning achieved performance and resource-awareness. The selection of a particular algorithm depends on the application requirements and the available resources of sensor nodes.

Future work includes the application of our resource-aware scheduling approach to different WSN applications, the implementation on our visual sensor network platforms [23], and the comparison of our approach with other variants of reinforcement learning methods.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported in part by the Erasmus Mundus Joint Doctorate in Interactive and Cognitive Environments, which is funded by the EACEA Agency of the European Commission under EMJD ICE FPA no. 2010-0012, and the EPiCS project funded by the European Union Seventh Framework Programme under Grant Agreement no. 257906.

References

[1] J. Ko, K. Klues, C. Richter et al., "Low power or high performance? A tradeoff whose time has come (and nearly gone)," in *Proceedings of European Conference on Wireless Sensor Networks*, pp. 98–114, 2012.

[2] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, no. 4, pp. 921–960, 2007.

[3] W. Schriebl, T. Winkler, A. Starzacher, and B. Rinner, "A pervasive smart camera network architecture applied for multi-camera object classification," in *Proceedings of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC '09)*, p. 8, Como, Italy, September 2009.

[4] A. Klausner, A. Tengg, and B. Rinner, "Distributed multilevel data fusion for networked embedded systems," *IEEE Journal on Selected Topics in Signal Processing*, vol. 2, no. 4, pp. 538–555, 2008.

[5] M. I. Khan and B. Rinner, "Resource coordination in wireless sensor networks by cooperative reinforcement learning," in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops*, pp. 895–900, March 2012.

[6] C. Frank and K. Römer, "Algorithms for generic role assignment in wireless sensor networks," in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*, pp. 230–242, 2005.

[7] Y. Tian, E. Ekici, and F. Özgüner, "Energy-constrained task mapping and scheduling in wireless sensor networks," in *Proceedings of the 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS '05)*, pp. 211–218, November 2005.

[8] M. Bramberger, M. Quaritsch, T. Winkler, B. Rinner, and H. Schwabach, "Integrating multi-camera tracking into a dynamic task allocation system for smart cameras," in *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS '05)*, pp. 474–479, Como, Italy, September 2005.

[9] H. Kopetz, *Real-Time Systems-Design Principles for Distributed Embedded Applications*, Springer, 2011.

[10] W. Guo, N. Xiong, H.-C. Chao, S. Hussain, and G. Chen, "Design and analysis of self-adapted task scheduling strategies in wireless sensor networks," *Sensors*, vol. 11, no. 7, pp. 6533–6554, 2011.

[11] S. Giannecchini, M. Caccamo, and C.-S. Shih, "Collaborative resource allocation in wireless sensor networks," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS '04)*, pp. 35–44, July 2004.

[12] B. Krishnamachari, S. Wicker, R. Béjar, and C. Fernàndez, "On the complexity of distributed self-configuration in wireless networks," *Journal of Telecommunication Systems*, vol. 22, no. 1–4, pp. 33–59, 2003.

[13] S. Dhanani, J. Arseneau, A. Weatherton, B. Caswell, N. Singh, and S. Patek, "A comparison of utility-based information management policies in sensor networks," in *IEEE Systems and Information Engineering Design Symposium*, pp. 84–89, Systems Technology Integration Lab of the Department of Systems and Information Engineering at the University of Virginia, Charlottesville, Va, USA, April 2006.

[14] K. Shah and M. Kumar, "Distributed independent reinforcement learning (DIRL) approach to resource management in wireless sensor networks," in *Proceedings of the IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems (MASS '07)*, pp. 1–9, October 2007.

[15] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[16] M. I. Khan and B. Rinner, "Resource coordination in wireless sensor networks by cooperative reinforcement learning," in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops*, pp. 871–877, March 2012.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, Mass, USA, 1998.

[18] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2003.

[19] K.-L. A. Yau, P. Komisarczuk, and P. D. Teal, "Reinforcement learning for context awareness and intelligence in wireless networks: review, new features and open issues," *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 253–267, 2012.

[20] J. Byers and G. Nasser, "Utility based decision making in wireless sensor networks," in *Proceedings of the Workshop on Mobile and Ad Hoc Networking and Computing*, pp. 143–144, 2000.

[21] U. A. Khan and B. Rinner, "Online learning of timeout policies for dynamic power management," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 4, p. 25, 2013.

[22] T. Abbes, S. Mohamed, and K. Bouabdellah, "Impact of model mobility in Ad Hoc routing protocols," *Computer Network and Information Security*, vol. 10, pp. 47–54, 2012.

[23] L. Esterle, P. R. Lewis, X. Yao, and B. Rinner, "Socio-economic vision graph generation and handover in distributed smart camera networks," *ACM Transactions on Sensor Networks*, vol. 10, no. 2, Article ID 2530001, 2014.