Khan *EURASIP Journal on Wireless Communications and Networking* (2016) 2016:10
DOI 10.1186/s13638-015-0515-y

EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

Resource-aware task scheduling by an adversarial bandit solver method in wireless sensor networks

Muhidul Islam Khan

Abstract

A wireless sensor network (WSN) is composed of a large number of tiny sensor nodes. Sensor nodes are very resource-constrained, since nodes are often battery-operated and energy is a scarce resource. In this paper, a resource-aware task scheduling (RATS) method is proposed with better performance/resource consumption trade-off in a WSN. Particularly, RATS exploits an adversarial bandit solver method called exponential weight for exploration and exploitation (Exp3) for target tracking application of WSN. The proposed RATS method is compared and evaluated with the existing scheduling methods exploiting online learning: distributed independent reinforcement learning (DIRL), reinforcement learning (RL), and cooperative reinforcement learning (CRL), in terms of the tracking quality/energy consumption trade-off in a target tracking application. The communication overhead and computational effort of these methods are also computed. Simulation results show that the proposed RATS outperforms the existing methods DIRL and RL in terms of achieved tracking performance.

Keywords: Wireless sensor networks, Task scheduling, Resource-awareness, Independent reinforcement learning, Cooperative reinforcement learning, Adversarial bandit solvers

1 Introduction

Wireless sensor networks (WSNs) [1] are an important and attractive platform for various pervasive applications like target tracking, area monitoring, routing, and in-network data aggregation. Resource awareness is an important issue for WSNs. Basically, battery power, memory, and processing functionality form the resource infrastructure. A WSN has its own resource and design constraints. Resource constraints include a limited energy, low bandwidth, limited processing capability of the central processing unit, limited storing capacity of the storage device, and short communication range. Design constraints are application-dependent and also depend on the environment being monitored. The environment acts as a major determinant regarding the size of the network, deployment strategy, and network topology. The number of sensor nodes or the size of the network changes based on the monitored environment. For example, in indoor environments, fewer nodes are needed to form a network in a limited space, whereas outdoor environments may require more sensor nodes to cover a huge unattended area. The deployment scheme also depends on the environment. Ad hoc deployment is preferred over a pre-planned deployment when the environment is not accessible and the network is composed of a vast number of nodes.

Battery power is the main resource constraint of a WSN. One of the major reasons of energy consumption for the communication in WSN is idle mode consumption. When there is no transmission/reception, sensor nodes consume some energy for listening and waiting for the information from the neighboring nodes. Over hearing is another source of energy consumption. Over hearing means that a node picks up packets that are destined for other nodes. Packet collision is another issue of energy consumption. Collided packets should be retransmitted which require extra effort in energy consumption. Protocol overhead is also a reason of energy consumption.

As a WSN is a resource-constrained network, there are challenges associated with task scheduling. For performing the application, sensor nodes execute some tasks. Task

Correspondence: muhit.islam.khan@gmail.com
BRAC University, Dhaka, Bangladesh

scheduling methods help to schedule the tasks in a way that the resource is optimized with the goal of maximizing the lifetime of the network. Sensor nodes consume some resources from the resource budget for each executed task. Scheduling can be performed online, offline, or periodically.

Sensor nodes pose strong energy limitations due to fixed battery operation [2–4]. Based on the application demand, sensor nodes need to execute a particular task at each time step. Every task execution consumes some energy from the available energy budget of the sensor node. The main goal is to improve the highest amount of performance while keeping the energy consumption low by exploiting online learning [5] for scheduling.

For example, in object tracking application, sensor nodes need to perform some tasks like sensing, transmitting, receiving, and sleeping over time steps. The performed task at each time step provides an impact on the overall tracking performance by a WSN. There is a trade-off between tracking performance and energy consumption for the tasks. If we perform tracking all the time, then it is possible to get higher tracking quality, but this is very energy consuming. It is necessary to schedule the tasks in a way that the energy consumption is optimized, and at the same time, a certain amount of tracking quality is maintained.

Since it is not possible to schedule the tasks a priori, online and energy-aware task scheduling is required. For determining the next task to execute, sensor nodes need to consider the available energy and the energy required for executing that task. Sensor nodes also need to consider the effect of executing the task on the application's overall performance.

In this paper, an online learning algorithm is proposed for the task scheduling in order to explore the trade-off between performance and energy consumption. A bandit solver method Exp3 (Exp3 denotes exponential weight for exploration and exploitation) is used [6]. Exp3 is an adversarial bandit solver method used for online task scheduling. This works by maintaining a list of weights for each task to perform. Using these weights, it decides randomly which task to take next and increases/decreases the relevant weights when a payoff is good or bad. Exp3 has an egalitarianism factor which tunes the desire to pick a task uniformly at random.

The proposed resource-aware task scheduling (RATS) method is based on a simulation study of the performance and energy consumption of a prototypical target tracking application. The balancing factor of the reward function is varied, and the number of nodes in the network and the randomness of moving targets to find out the tracking quality/energy consumption trade-off are also varied. The average execution time and communication overhead for distributed independent reinforcement learning (DIRL) [7], reinforcement learning (RL) [8], cooperative reinforcement learning (CRL) [9], and Exp3 are also calculated.

The main contribution of this paper is to propose a method for RATS and to perform the evaluation with the existing methods. The proposed RATS approach also considers the cooperation where each node shares local observations of object trajectories with the neighboring nodes. This cooperation helps to improve the tracking performance of our considered tracking application.

The rest of this paper is organized as follows. Section 2 discusses related works, and Section 3 introduces the network model. Section 4 describes the underlying system model for task scheduling based on online learning. In Section 6, we present the proposed method for resource-aware task scheduling. Section 7 presents the experimental setup and discusses the simulation results for a target tracking application. Section 8 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 use of resources [10, 11]. Cooperative behavior among sensor nodes can be very helpful to schedule the tasks in a way that the energy consumption is optimized and also a considerable performance is maintained. Most of the existing methods of task scheduling do not provide online scheduling of tasks. Most of them rather consider static task allocation instead of focusing on distributed task scheduling. The main difference between task allocation and distributed task scheduling is that task allocation deals with the problem of finding a set of task assignments on a sensor network that minimizes an objective function such as total execution time [12, 13]. On the other hand, in a task scheduling problem, the objective is to determine the "best" order of task execution for each sensor node. Each sensor node has to execute a particular task at each time step in order to perform the application, and each node determines the next task to execute based on the observed application behavior and available resources. The following subsections describe some task scheduling methods in WSN.

2.1 Self-adaptive task allocation

Guo et al. [14] 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 consider neither distributed task scheduling nor the trade-off among energy consumption and performance.

Xu et al. [15] propose a novel hierarchical data aggregation method using compressive sensing which combines a hierarchical network configuration. Their key idea is

to set multiple compression thresholds adaptively based on cluster sizes at different levels of the data aggregation tree to optimize the amount of data transmitted. The advantages of the proposed model in terms of the total amount of data transmitted and data compression ratio are analytically verified. Moreover, they formulate a new energy model by factoring in both processor and radio energy consumption into the cost, especially the computation cost incurred in relatively complex algorithms.

2.2 Collaborative resource allocation

Giannecchini et al. [16] 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 does also not explicitly consider energy consumption.

Meng et al. [17] argue that by carefully considering spatial reusability of the wireless communication media, they can tremendously improve the end-to-end throughput in multi-hop wireless networks. To support their argument, they propose spatial reusability-aware single-path routing (SASR) and any-path routing (SAAR) protocols and compare them with existing single-path routing and any-path routing protocols, respectively.

2.3 Rule-based method

Frank and Römer [10] propose an algorithm for generic task allocation in WSNs. 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 pre-installed on the nodes. A role assignment algorithm takes into account the rules and node properties, which may trigger execution and network data aggregation. This generic role assignment approach does consider the energy consumption but not the ordering of tasks to sensor nodes.

2.4 Constraint satisfaction-based method

Krishnamachari and Wicker [18] examine the channel utilization as a resource management problem by a distributed constraint satisfaction method. They consider a WSN of n nodes placed randomly in a square area with a uniform, independent distribution. This work tests three self-configuration tasks in WSNs: 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 0 to 1.

Busch et al. [19] propose a classic optimization problem in network routing which is to minimize $C + D$, where C is the maximum edge congestion and D is the maximum path length (also known as dilation). The problem of computing the optimal is NP-complete. They study routing games in general networks where each player i selfishly selects a path that minimizes the sum of congestion and dilation of the player's path.

These constraint satisfaction approaches neither address mapping of tasks to sensor nodes nor discuss the resource consumption/performance trade-off.

2.5 Utility-based method

Dhanani et al. [20] 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 a 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. Peng et al. [21] propose a reliable multi-cast protocol, called CodePipe, with energy efficiency, high throughput and fairness in lossy wireless networks. Building upon opportunistic routing and random linear network coding, CodePipe can not only eliminate coordination between nodes but also improve the multi-cast throughput significantly by exploiting both intra-batch and inter-batch coding opportunities.

These approaches do not address the task scheduling to improve the resource consumption/performance trade-off.

2.6 Reinforcement learning-based method

Reinforcement learning helps to enable applications with inherent support for efficient resource/task management. It is the process by which an agent improves task scheduling according to previously learned behavior. It does not need a model of its environment and can be used online. It is simple and demands minimal computational resources.

Shah and Kumar [7] consider Q learning as reinforcement learning for the task management. They describe a distributed independent reinforcement learning (DIRL) approach for resource management, which forms an important component of any application including initial

sensor selection and task allocation as well as run-time adaptation of allocated resources to tasks. Here, the optimization parameters are energy, bandwidth, network lifetime, etc. DIRL allows each individual sensor node to self-schedule its tasks and allocate its resources by learning their usefulness in any given state while honoring application-defined constraints and maximizing the total amount of reward over time.

Khan and Rinner [8] apply reinforcement learning (RL) for online task scheduling. They use cooperative reinforcement learning for task scheduling. They introduce cooperation among neighboring nodes with the local information of each node. This cooperation helps to provide better performance. Cooperative Q learning is a reinforcement learning approach to learn the usefulness of some tasks over time in a particular environment. They consider the WSN as a multi-agent system. The nodes correspond to agents in the multi-agent reinforcement learning. The world surrounding the sensor nodes forms the environment. Tasks are considered as activities for the sensor nodes at each time step such as transmit, receive, sleep, and sense. States are formed by a set of system variables such as object in the field of view (FOV) of sensor nodes, required energy for a specific action, and data to transmit. A reward value provides some positive or negative feedback for performing a task at each time step. Value functions define what is good for an agent over the long run described by reward function and some parameters.

Khan and Rinner [9] apply cooperative reinforcement learning (CRL) for online task scheduling. They use State-Action-Reward-State-Action ($SARSA(\lambda)$) [22] learning and introduced cooperation among neighboring sensor nodes to further improve the task scheduling.

The proposed RATS method applies Exp3 which does not need any statistical assumptions like stochastic bandit solvers. Exp3 is an online bandit solver in which an adversary, rather than a well-behaved stochastic process, has complete control over the payoffs/rewards [6].

DIRL, RL, CRL, and Exp3 are compared for the task scheduling in a target tracking application and analyzed for the performance in terms of tracking quality/energy consumption trade-off.

3 Network model

Before describing the problem formally, terms like resource consumption and tracking quality need to be defined. In WSNs, resource consumption happens because of performing the various tasks needed for the application. Each task consumes an amount of energy from the fixed energy budget of the sensor nodes. Typically, tracking quality in a target tracking application of a WSN is defined as the accuracy of target location estimation provided by the network.

In the proposed approach, a WSN is composed of N nodes represented by the set $\hat{N} = \{n_1, \ldots, n_N\}$. Each node has a known position (u_i, v_i) and a given sensing coverage range which is simply modeled by 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 $\text{ngh}(n_i)$. The available resources of node n_i are modeled by a scalar E_i. The battery power of sensor nodes is considered as resource. A set of tasks is considered to perform over time steps. Each task consumes some battery power from the energy budget of the sensor nodes. A set of static values for the energy consumption of tasks is considered. These values are assigned based on the energy demands of the task. A higher value is set for the tasks which need higher energy consumption.

The WSN application is composed of A tasks (or actions) represented by the set $\hat{A} = \{a_1, \ldots, a_A\}$. Once a task is started at a specific node, it executes for a specific (short) period of time and terminates afterwards. Each task execution on a specific node n_i requires some resources $\tilde{E}_j$ and contributes to the overall application performance P. Thus, the execution of task a_j on node n_i is only feasible if $E_i \geq \tilde{E}_j$. The overall performance P is represented by an application-specific metric. 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. Non-preemptive scheduling is considered based on our proposed model. Figure 1 shows our considered WSN model components.

Table 1 shows the notations and meanings used to represent the network model. The task scheduling approach is demonstrated using a target tracking application. A sensor network may consist of a variable number of nodes. The sensing region of each node is called the field of view (FOV). Every node aims to detect and track all targets in the FOV. If the sensor nodes would perform tracking all the time, then this would result in the best tracking performance. But executing target tracking all the time is energy demanding. Thus, a task should only be executed when necessary and sufficient for tracking performance. Sensor nodes can cooperate with each other by informing neighboring nodes about "approaching" targets. Neighboring nodes can therefore become aware of approaching targets.

The objective function is defined in a way that it is possible to trade the application performance and the required energy consumption by a balancing factor. The ultimate objective of the problem is to determine the order of tasks on each node such that the overall performance is maximized while the resource consumption

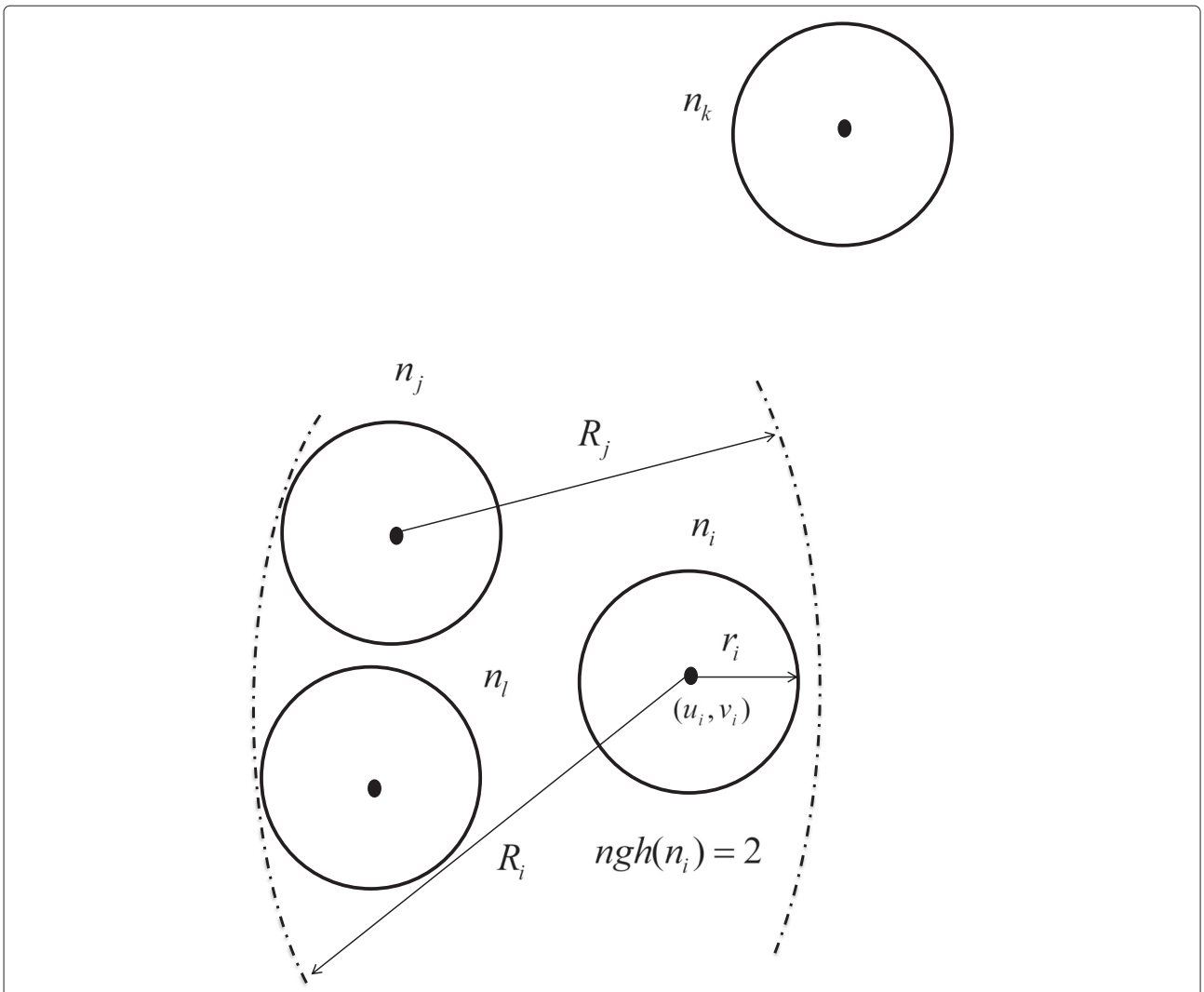

Fig. 1 WSN model components. There are four nodes $n_i, n_j \ldots n_k$, and n_l. R_i is the communication range, r_i is the sensing range, and (u_i, v_i) is the position of the node n_i. Number of neighbors of the node n_i, $ngh(n_i) = 2$

is minimized. Naturally, these are conflicting optimization criteria, so there is no single best solution. The set of non-dominating solutions for such a multi-criteria problem can be typically represented by a Pareto front.

Table 1 Notations used to represent the network model

Notation	Meaning
$\hat{N}$	Set of nodes WSN consists of
(u_i, v_i)	Known position of a node
r_i	Sensing range of node i
R_i	Communication range of node i
$\hat{A}$	Set of available actions
E_i	Available resources of node i
$\tilde{E}_j$	Required resources for task execution
P	Overall performance

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. We consider the set of tasks, set of states, and the reward function as considered in [23]. Figure 2 depicts the scheduling framework where its key components can be described as follows:

- *Agent*: Each sensor node embeds an agent which is responsible for executing the online learning algorithm.
- *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.
- *Action*: An agent's action is the currently executed application task on the sensor node. At the end of

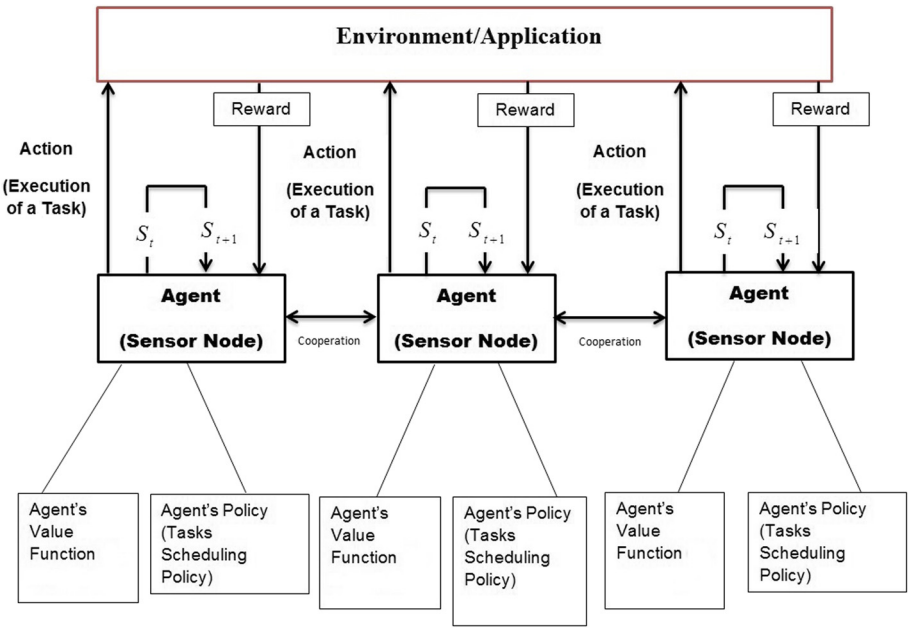

Fig. 2 General framework for task scheduling using online learning

each time period t_i, each node triggers the scheduler to determine the next action to execute.

- *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.
- *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.
- *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 [7].
- *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 trade-off between energy consumption and tracking performance.
- *Cooperation*: Information exchange is considered among neighboring nodes as cooperation. The received information may influence the application's state of sensor nodes.

5 Existing methods for task scheduling

Existing methods DIRL, RL, and CRL are described below. Each method is described briefly with the learning mechanism and considered set of states and tasks.

5.1 DIRL

Shah and Kumar [7] use distributed independent Q learning (DIRL) as reinforcement learning. The advantage of using independent learning is that no communication is required for coordination between sensor nodes and each node selfishly tries to maximize its own rewards. In Q learning, every agent needs to maintain a Q matrix for the value functions. Initially, all entries of the Q matrix are 0 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

$$Q_{t+1}(s_t, a_t) = (1-\alpha)Q_t(s_t, a_t)+\alpha(r_{t+1}(s_{t+1})+\gamma V_t(s_{t+1})), \quad (1)$$

$$V_{t+1}(s_t) = \max_{a \in A} Q_{t+1}(s_t, a) \quad (2)$$

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.

Algorithm 1 Q learning for task scheduling.

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 (Eq. 1)
7: Calculate the value function for the executed action (Eq. 2)
8: Shift to next state based on the executed action
9: **end while**

5.2 RL

Khan and Rinner [8] propose cooperative Q learning (RL) where every agent needs to maintain a Q matrix for the value functions like independent Q learning. Initially, all entries of the Q matrix are 0 and the nodes or agents may be in any state. Based on the application-defined variable or system variables, the system goes to a particular state. Then it performs an action which depends on the status of the nodes (example: For transmit action, a node must have residual energy which is greater than transmission cost). It calculates the Q value for this (state, task) pair with the immediate reward.

$$Q_{t+1}(s_t, a_t) = (1 - \alpha) Q_t(s_t, a_t) + \alpha (r_{t+1}(s_{t+1}) + \gamma \sum f V_t(s_{t+1})) \tag{3}$$

$$V_{t+1}(s_t) = \max_{a \in A} Q_{t+1}(s_t, a) \tag{4}$$

where $Q_{t+1}(s_t, a_t)$ means the update of Q value at time $t+1$, after executing the action a at time step t. r_{t+1} means the immediate reward after executing the action a at time t. V_t is the value function at time t. V_{t+1} is 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. γ is the discount factor which can be set to a value in $[0, 1]$. The higher the value, the greater the agent relies on future reward 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 set close to 1, the agent gives more priority to the previously learned utility value.

f is the weight factor [24] for the neighbors of agent i and can be defined as follows:

$$f = \frac{1}{\text{ngh}(n_i)} \quad \text{if ngh}(n_i) \neq 0 \tag{5}$$

$$f = 1 \quad \text{otherwise.} \tag{6}$$

The algorithm can be stated as follows:

Algorithm 2 Q learning for task scheduling.

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 not equal to 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 (Eq. 3)
7: Calculate the value function for the executed action (Eq. 4)
8: Send the value function to the neighbors
9: Shift to next state based on the executed action
10: **end while**

5.3 CRL

Khan and Rinner [9] apply $SARSA(\lambda)$ (CRL), also referred to as State-Action-Reward-State-Action, which is an iterative algorithm that approximates the optimal solution. $SARSA(\lambda)$ [22] 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 $SARSA(\lambda)$ algorithm has the following updating rule for the Q values:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s, a) + \alpha \delta_t e_t(s_t, a_t) \tag{7}$$

for all s, a.

In Eq. 7, $\alpha \in [0, 1]$ is the learning rate which decreases with time. δ_t is the temporal difference error which is calculated by following rule

$$\delta_t = r_{t+1} + \gamma_1 f Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{8}$$

In Eq. 8, γ_1 is a discount factor which varies from 0 to 1. The higher the value, 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 [24] for the neighbors of agent i and can be defined as follows:

$$f = \frac{1}{\text{ngh}(n_i)} \quad \text{if ngh}(n_i) \neq 0 \tag{9}$$

$$f = 1 \quad \text{otherwise.} \tag{10}$$

An important aspect of an RL framework is the trade-off between exploration and exploitation [25]. 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.

A simple heuristic is used where exploration probability at any point of time is given by

$$\epsilon = \min(\epsilon_{\max}, \epsilon_{\min} + k * (S_{\max} - S)/S_{\max}) \tag{11}$$

where $\epsilon_{\max}$ and $\epsilon_{\min}$ denote upper and lower boundaries for the exploration factor, respectively. $S_{\max}$ represents maximum number of states which is three in our work, and S represents current number of states already known. At each time step, the system 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).

$SARSA(\lambda)$ improves learning through eligibility traces. $e_t(s, a)$ is the eligibility traces in Eq. 7. 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 algorithm can be stated as follows:

Algorithm 3 $SARSA(\lambda)$ learning algorithm for target tracking application.

```
Initialize Q(s,a) = 0 and e(s,a) = 0
while Residual energy is not equal to zero do
    Determine current state s by application variables
    Select an action a, using policy
    Execute the selected action
    Calculate reward for the executed action (Eq. 38)
    Update the learning rate (Eq. 14)
    Calculate the temporal difference error (Eq. 8)
    Update the eligibility traces (Eq. 13)
    Update the Q-value (Eq. 7)
    Shift to next state based on the executed action
end while
```

The eligibility trace is updated by the following rule:

$$e_t(s_t, a_t) = \gamma_2 \lambda e_{t-1}(s_t, a_t) + 1 \text{ if } s_t \in s \text{ and } a_t \in a \tag{12}$$

$$e_t(s_t, a_t) = \gamma_2 \lambda e_{t-1}(s_t, a_t) \quad \text{otherwise.} \tag{13}$$

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

$$\alpha = \frac{\zeta}{\text{visited}(s, a)} \tag{14}$$

where ζ is a positive constant and visited(s, a) represents the visited state-action pairs so far [26].

6 Proposed method for task scheduling

Following set of actions, set of states and reward function are considered for the proposed RATS.

6.1 Set of actions

The following actions are considered in our target tracking application:

1. *Detect_Targets*: This function scans the field of view (FOV) and returns the number of detected targets in the FOV.
2. *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.
3. *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.
4. *Predict_Trajectory*: This function predicts the velocity of the trajectory. A simple approach is to use the two most recent target positions, i.e., (x_t, y_t) at time t_t and (x_{t-1}, y_{t-1}) at t_{t-1}. Then the constant target's speed can be estimated as
$$v = \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2}/(t_t - t_{t-1}) \tag{15}$$
5. *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
$$\tilde{t}_i = D_{P_i P_j}/v \tag{16}$$
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. Fig. 3). v is the estimated velocity as calculated by Eq. 15.

6. *Goto_Sleep*: This function shuts down the sensor node for a single time period. It consumes the least amount of energy of all available actions.

The advanced trajectory prediction and intersection are considered for these methods. Inputs for this prediction task are the last few tracked positions of the target. Here, the last six tracked positions of the target are considered based on simulation studies. The trajectory is linearized given by the last six tracked positions of the target considering the constant speed and direction. The speed is calculated by Eq. 15.

Suppose $(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)$ are tracked positions of the moving object inside the FOV of the sensor node at time steps $t_1, t_2 \ldots t_n$.

The trajectory can be predicted by the regression line [27] in Eq. 17:

$$y = bx + a + \epsilon \tag{17}$$

where b is the slope, a is the intercept, and ϵ is the residual or error for the calculation.

So residual, ϵ, can be calculated by following

$$\epsilon_i = y_i - bx_i - a \tag{18}$$

where $i = 1, 2, 3, \ldots, n$.

If the squares of the residuals of all the points from the line are summed up, what we get is a measure of the fitness of the line. The aim is to minimize this value.

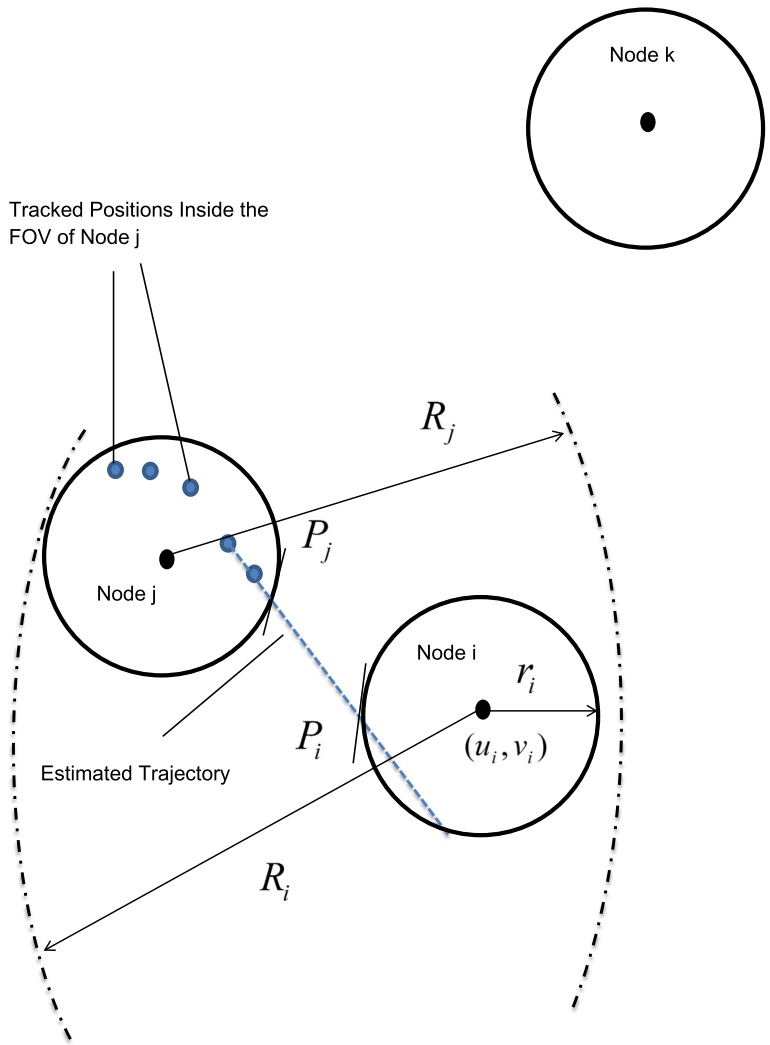

Fig. 3 Target prediction and intersection. Node *j* estimates the target trajectory and sends the trajectory information to its neighbors. Node *i* checks whether the predicted trajectory intersects its FOV and computes the expected arrival time

So, the square of the residual is as follows:

$$\epsilon_i^2 = (y_i - bx_i - a)^2 \tag{19}$$

To calculate the sum of square residuals, all the individual square residuals are added together as follows:

$$J = \sum_{i=1}^{n} (y_i - bx_i - a)^2 \tag{20}$$

where J is the sum of square residuals and n is the number of considered points.

J in Eq. 20 needs to be minimized. The minimum value for J has to occur when the first derivative is 0. The partial derivatives for J are with respect to the two parameters of the regression line b and a. To get the minimum, it needs to assign 0 [28].

$$\frac{\partial J}{\partial b} = \sum_{i=1}^{n} 2(y_i - bx_i - a)(-x_i) = 0 \tag{21}$$

$$\frac{\partial J}{\partial a} = \sum_{i=1}^{n} 2(y_i - bx_i - a)(-1) = 0 \tag{22}$$

Equations 21 and 22 can be shuffled and divided as

$$\sum_{i=1}^{n} bx_i + \sum_{i=1}^{n} a = \sum_{i=1}^{n} y_i \tag{23}$$

$$\sum_{i=1}^{n} bx_i^2 + \sum_{i=1}^{n} ax_i = \sum_{i=1}^{n} x_i y_i \tag{24}$$

Some constants can be pulled out in front of the summations. The $\sum_{i=1}^{n} a$ can be written as na in Eq. 23. We take the values of unknown parameters a and b using Eqs. 23 and 24. These give two equations as follows:

$$b \sum_{i=1}^{n} x_i + \text{na} = \sum_{i=1}^{n} y_i \tag{25}$$

$$b \sum_{i=1}^{n} x_i^2 + a \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i \tag{26}$$

Now, from Eqs. 25 and 26, some simple substitutions between the two equations are obtained as follows:

$$a = \frac{\sum y}{n} - b\frac{\sum x}{n} \tag{27}$$

$$b = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2} \tag{28}$$

These formulas in Eqs. 27 and 28 do not tell us how precise the estimates are. That is, how much the estimators a and b can deviate from the "true" values of a and b. It can be solved by confidence intervals.

Using Student's t-distribution with $(n-2)$ degrees of freedom [29], a confidence interval can be constructed for a and b as follows:

$$\hat{b} \in \left[b - s_b t_{n-2}^*, b + s_b t_{n-2}^*\right] \tag{29}$$

$$\hat{a} \in \left[a - s_a t_{n-2}^*, a + s_a t_{n-2}^*\right] \tag{30}$$

where $\hat{a}$ and $\hat{b}$ are the new estimated values of a and b. t_{n-2}^* is the $(1-\tau/2)$-th quantile of the t_{n-2} distribution. For example, if $\tau = 0.05$, the confidence level becomes 95 %. s_a and s_b are the standard deviations as follows:

$$s_b = \sqrt{\frac{\frac{1}{n-2}\sum_{i=1}^{n} \varepsilon_i^{\,2}}{\sum_{i=1}^{n}(x_i - \bar{x})^2}} \tag{31}$$

$$\begin{aligned} s_a &= s_b \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \\ &= \sqrt{\frac{1}{n(n-2)}\left(\sum_{i=1}^{n} \varepsilon_i^{\,2}\right)\frac{\sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}} \end{aligned} \tag{32}$$

where $\bar{x}$ is the average of the x values.

In Fig. 4, some tracked positions of the target are observed which is denoted by the "black" dots. At first, the regression line is predicted and the middle line is obtained. Then the confidence band is calculated which gives two other lines.

For the intersection with the circles, the line as follows is considered:

$$y = bx + a \tag{33}$$

where b is the slope and a is the intercept.

The line given by Eq. 33 intersects a circle (sensing range is considered as a circle) given by Eq. 34:

$$(x - u_1)^2 + (y - v_1)^2 = r_1^2 \tag{34}$$

where (u_1, v_1) is the center and r_1 is the radius of the circle.

Substituting the value of Eq. 33 in Eq. 34 gives the following:

$$(x - u_1)^2 + ((bx + a) - v_1)^2 = r_1^2 \tag{35}$$

Simply expanding Eq. 35 by algebraic formula gives a quadratic equation of x and can be solved using the quadratic formula.

After solving the quadratic equation, we get the values of x and y.

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \tag{36}$$

if $B^2 - 4AC < 0$, then the line misses the circle. If $B^2 - 4AC = 0$, then the line is tangent to the circle. If $B^2 - 4AC > 0$, then the line meets the circle in two distinct points.

x can be substituted in Eq. 33 from Eq. 36 to get the y values:

$$y = b\left(\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}\right) + a \tag{37}$$

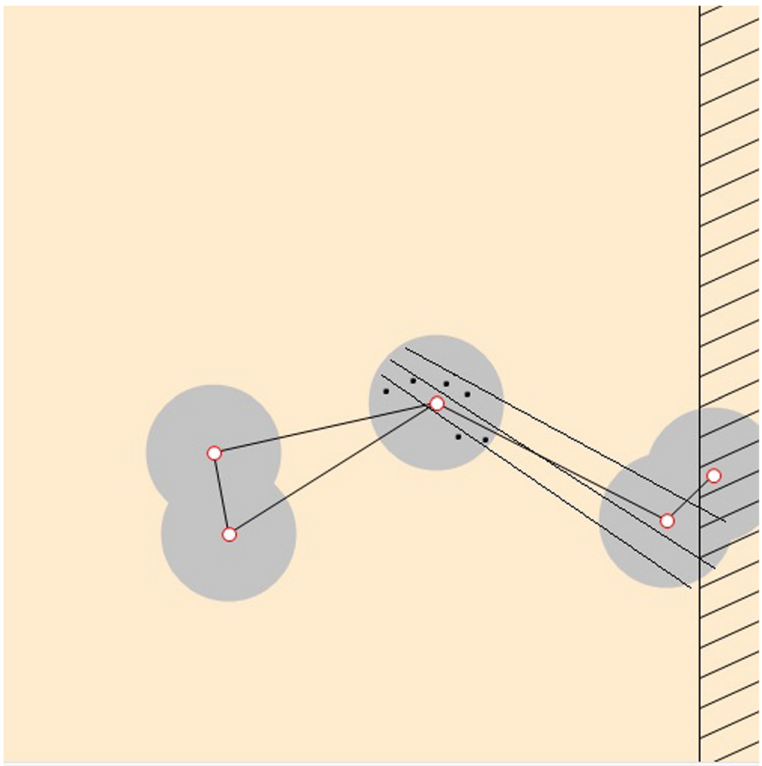

Fig. 4 Trajectory prediction and intersection. *Black dots* denote the tracked positions of a target. The *middle line* is drawn based on linear regression. The *other two lines* are drawn by confidence interval

6.2 Set of states

The application is abstracted by three states at every node.

- *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. Fig. 5), 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.
- *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.
- *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, i.e., whether to focus more on tracking performance or energy consumption. The states can be identified by two application variables, i.e., 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 a necessary task for 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

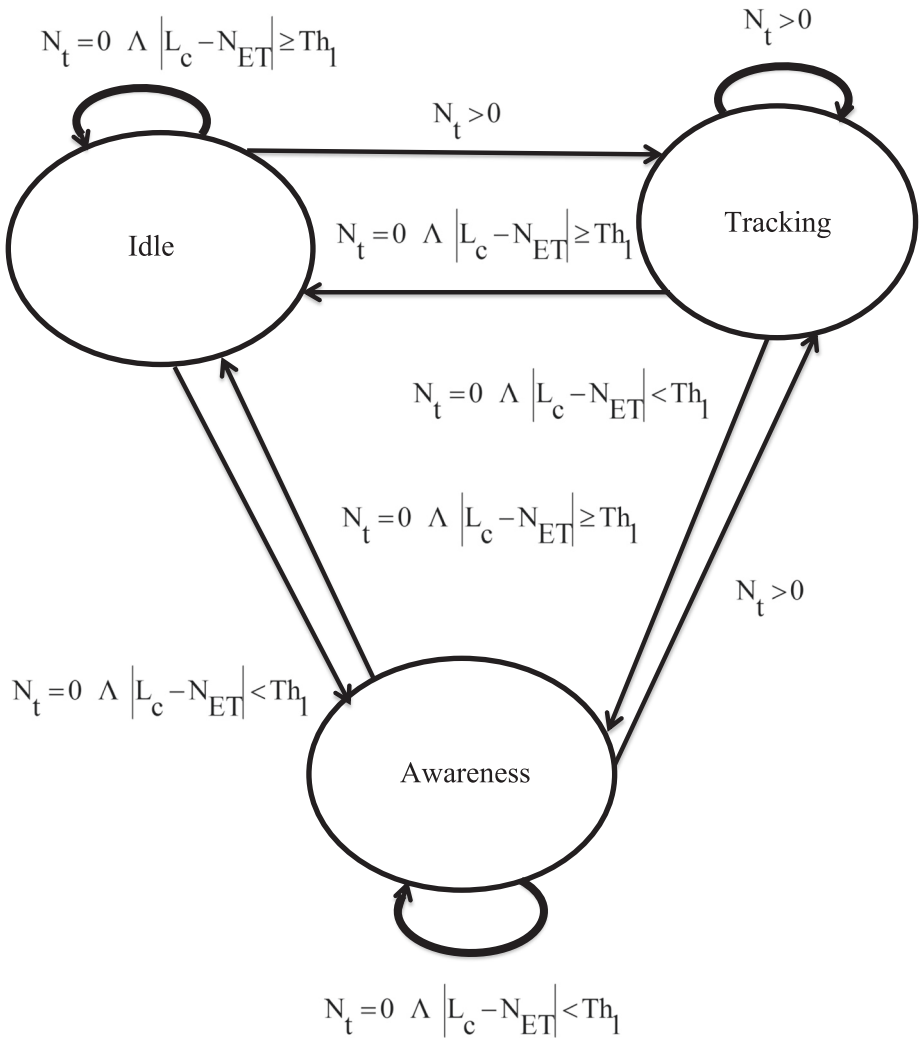

Fig. 5 State transition diagram. States change according to the value of two application variables N_t and N_{ET}. L_c represents the local clock value, and Th_1 is a time threshold

quality. But this situation could provide better energy efficiency, since the *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 trade-off.

Figure 5 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}.

6.3 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, it is simplified by merging energy consumption and system performance using a balancing parameter. In detail, the reward function in our algorithm is defined as

$$r = \beta(E_i/E_{\max}) + (1-\beta)(P_t/P) \tag{38}$$

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 the sensor node, and P is the number of all possible detected target's positions in the FOV. These two parameters 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.

6.4 Proposed method

The classical adversarial algorithm Exp3 (exponential-weight algorithm for exploration and exploitation) is used for task scheduling [6].

The algorithm can be stated as follows:

Algorithm 4 Task Scheduling by Bandit Solver Exp3.

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 not equal to 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 (Eq. 41)
8: Update the weights (Eq. 40)
9: Calculate the updated probability distribution (Eq. 39)
10: Shift to next state based on the executed action
11: **end while**

Exp3 has a parameter κ which 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 $P_{1,t}, P_{2,t}, \ldots, P_{A,t}$. The distribution can be calculated by the following equation:

$$P_{j,t+1} = (1-\kappa)\frac{w_{a,t}}{\sum_{j=1}^{A} w_{j,t}} + \frac{\kappa}{A}, j = 1, 2, \ldots, A \quad (39)$$

where $w_{a,t}$ is the weight associated with the action a at time 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 in the uniform distribution is done to make sure that the algorithm tries out all actions A and gets good estimates of the rewards for each action.

Weight for each action can be calculated by following equation

$$w_{a,t} = w_{a,t-1} e^{\kappa r_{t+1}} \quad (40)$$

where r_{t+1} is the reward after executing the action a.

Reward can be calculated by following equation

$$r_{t+1} = \frac{r_t}{P_{a,t}} \quad (41)$$

where $P_{a,t}$ is the calculated probability distribution for the action a by Eq. 39.

Exp3 works by maintaining a list of weights w_i by Eq. 40 for each of the actions, using these weights to decide which action to take next based on a probability distribution P_t, and 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.

7 Experimental results and evaluation

7.1 Simulation environment

The proposed method is implemented and evaluated with other task scheduling methods using a WSN multi-target 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 the evaluation scenario, the sensor nodes are uniformly distributed in a 2D rectangular area. A given number of sensor nodes are placed randomly in this area which can result in partially overlapping FOVs of the nodes. However, placement of nodes on the same position is avoided. Before deploying the network, the network parameters should be configured using the configuration sliders.

The following network parameters can be configured by our simulator.

- Network size: Network size means the number of nodes in the network. In the current settings of the simulator, the number of sensor nodes can be varied between [3, 40].
- Sensor radius: Sensor radius is the sensing range of the sensors in the network. Sensor radius can be varied between [1, 50].
- Transmission radius: Transmission radius is the maximum distance within two sensor nodes communicating with each other. If it is set to a high value, nodes on the opposite side of the rectangular area may be able to reach each other. If it is set to a low value, nodes must be very close to communicate with each other. Transmission radius can be varied between [1, 50].

The network is displayed on the simulation environment as a set of red circles surrounded by gray circles. The red circles denote the sensor nodes, and the gray circles denote the sensing range of the nodes. Each node is connected to nearby nodes by black lines which represent the communication links. When a message is being exchanged, it appears as red. The color in the center of the red circle represents the battery status of the node, which gradually shifts from white to black. White color denotes the nodes with full power, and black color denotes the nodes with no power. When a node loses all power, the node becomes completely black. The gray area of the node shrinks and disappears. All of the communication links associated with the node disappear as well.

Targets move around in the area based on a Gauss-Markov mobility model [30]. The Gauss-Markov mobility model was designed to adapt to different levels of randomness via tuning parameters. Initially, each mobile target is assigned with a current speed and direction. At each time step t, the movement parameters of each target are updated based on the following rule:

$$S_t = \eta S_{t-1} + (1-\eta)S + \sqrt{1-\eta^2}S^G_{t-1} \tag{42}$$

$$D_t = \eta D_{t-1} + (1-\eta)D + \sqrt{1-\eta^2}D^G_{t-1} \tag{43}$$

where S_t and D_t are the current speed and direction of the target at time t, respectively. S and D are constants representing the mean value of speed and direction, respectively. S^G_{t-1} and D^G_{t-1} 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 $\eta = 0$, and linear motion is obtained if $\eta = 1$. At each time t, the target's position is given by the following equations:

$$x_t = x_{t-1} + S_{t-1}\cos(D_{t-1}) \tag{44}$$

$$y_t = y_{t-1} + S_{t-1}\sin(D_{t-1}) \tag{45}$$

7.2 Settings of parameters

In the 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 2 shows the energy consumption for the execution of each action. Sending messages over two hops consumes energy on both the sender and relay nodes. To simplify the energy consumption at the network level, only the energy consumption to ten units on the sending node only is aggregated. The egalitarianism factor $\kappa = 0.5$ is set for Exp3. The sensing radius is considered as $r_i = 5$, and the communication radius is set as $R_i = 8$. These fixed values are set for the parameters based on simulation studies. For each simulation run, the achieved tracking quality and energy consumption are aggregated and normalized to $[0, 1]$.

Table 2 Energy consumption of the individual actions

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

7.3 Performed experiments

For the evaluation, the following four experiments are performed with the following assumptions of parameters.

1. To find out the trade-off between tracking quality and energy consumption, the balancing factor β of the reward function is set between $[0.1, 0.9]$ in 0.1 steps, keeping the randomness of moving target as $\eta = 0.5$, setting the egalitarianism factor of Exp3 as $\kappa = 0.5$, and fixing the topology to five nodes.
2. The network size is varied to check the trade-off between tracking quality and energy consumption. Three different topologies consisting of 5, 10, and 20 sensor nodes are considered 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. The balancing factor is set $\beta = 0.5$ and the randomness of the mobility model $\eta = 0.5$ which is a constant for this experiment.
3. Randomness of moving targets η is set 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 setting the balancing factor $\beta = 0.5$ and fixing the topology to five nodes.
4. DIRL, RL, CRL, and Exp3 are evaluated in terms of average execution time and average communication effort. These values are measured from 20 iterations and represent the mean execution times and the mean of *Send_Message* task executions.

7.4 Discussion

Figure 6 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 trade-off for DIRL, RL, CRL, and Exp3 by varying the balancing factor β between [0.1, 0.9] in 0.1 steps. It is observed that CRL and Exp3 provide similar results, i.e., the corresponding data points are closely co-located. RL is energy-aware but is not able to achieve high tracking quality. DIRL achieves the most energy awareness but provides the least tracking quality.

Figure 7 shows the results of the 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, i.e., the CRL and Exp3 achieve almost similar results in terms of tracking quality/energy consumption trade-off and DIRL shows less tracking performance with the higher energy efficiency.

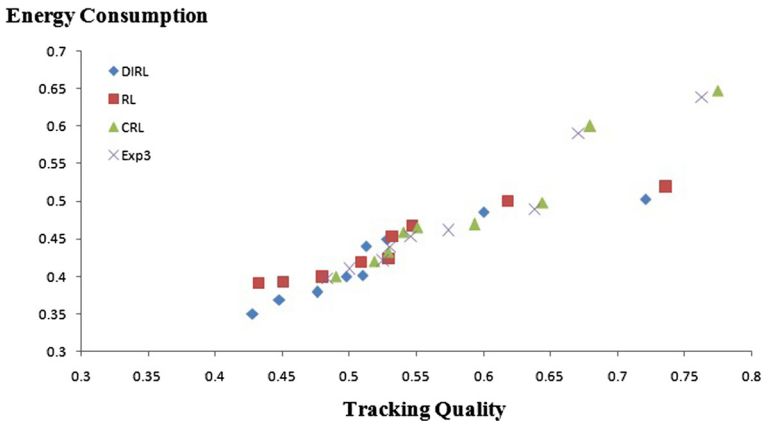

Fig. 6 Tracking quality/energy consumption trade-off for DIRL, RL, CRL, and Exp3 by varying the balancing factor of the reward function β

Figures 8, 9 and 10 show the results of our third experiment. In this experiment, each data point represents the average of normalized tracking quality and the 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 and DIRL in terms of achieved tracking performance. It can be seen that for lower randomness, $\eta = 0.5$, 0.7, and 0.9, RL and Exp3 show very close results for tracking performance. But for higher randomness, $\eta = 0.1$, 0.15, and 0.2, DIRL gives poor performance with regard to tracking performance.

Table 3 shows the comparison of DIRL, RL, CRL, and Exp3 in terms of average execution time and average communication effort. These values are derived from 20 iterations and represent the mean execution times and the mean of *Send_message* task executions. It can be seen that DIRL and RL are resource-aware in terms of execution time and communication effort. Exp3 requires 25 % more and CRL requires 86 % more execution time. The communication overhead is similar for both Exp3 and CRL.

8 Conclusions

In this paper, an adversarial bandit solver is applied based on online learning algorithm for resource-aware task scheduling in WSN. The performance of our proposed online task scheduling method is evaluated with other existing task scheduling methods based on the three learning algorithms: DIRL, RL, and CRL. Evaluation results show that our proposed RATS method provides better performance in terms of tracking quality. DIRL shows the best energy efficiency but provides poor results in terms of tracking quality. The proposed RATS and CRL

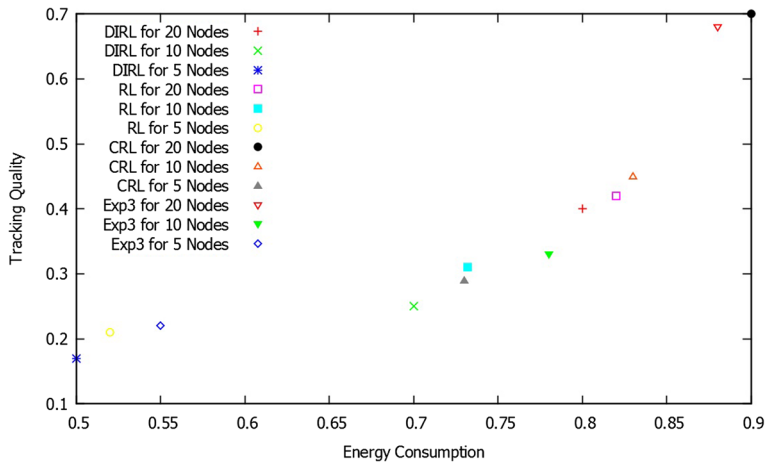

Fig. 7 Tracking quality/energy consumption trade-off for DIRL, RL, CRL, and Exp3 by varying the network size

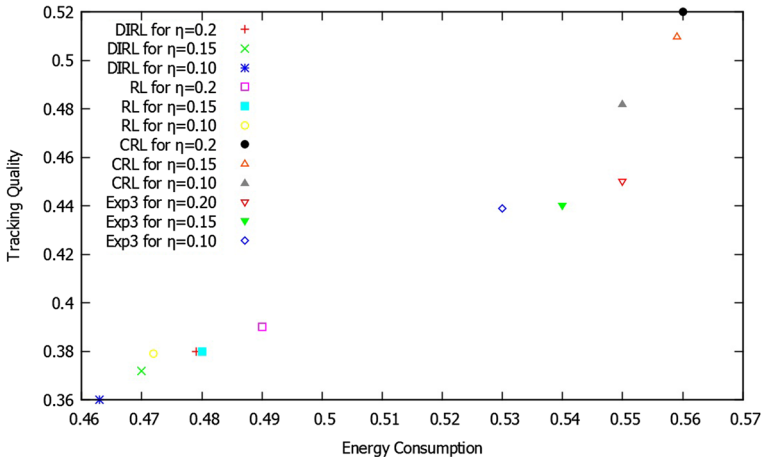

Fig. 8 Tracking quality/energy consumption trade-off for DIRL, RL, CRL, and Exp3 with different randomness of target movements, $\eta = 0.10, 0.15,$ and 0.20

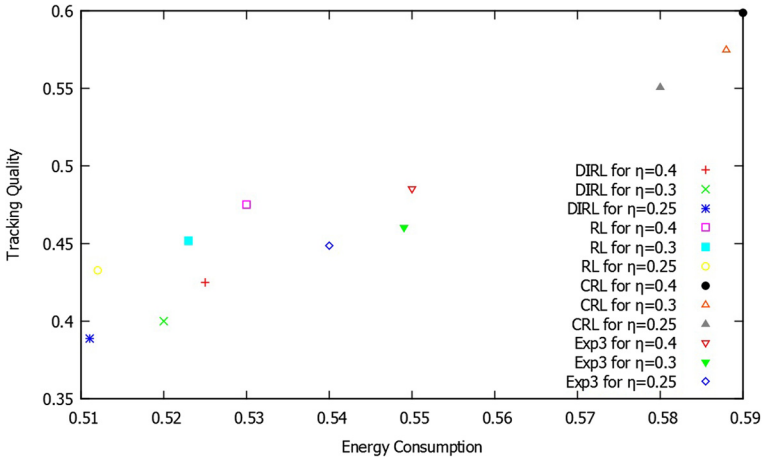

Fig. 9 Tracking quality/energy consumption trade-off for DIRL, RL, CRL, and Exp3 with different randomness of target movements, $\eta = 0.25, 0.30,$ and 0.40

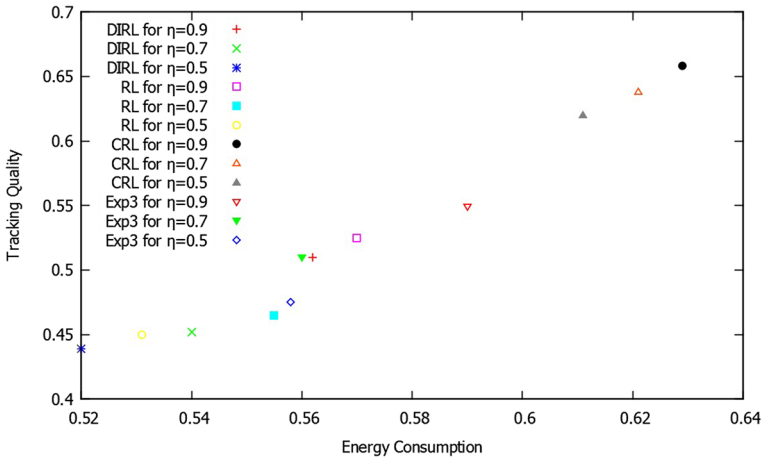

Fig. 10 Tracking quality/energy consumption trade-off for DIRL, RL, CRL, and Exp3 with different randomness of target movements, $\eta = 0.50, 0.70,$ and 0.90

Table 3 Comparison of average execution time and average number of transferred messages (based on 20 iterations)

	Avg. execution time (s)	Avg. comm. effort
DIRL	0.030	0
RL	0.036	0
CRL	0.067	29
Exp3	0.045	27

show almost similar results in terms of tracking quality-energy consumption trade-off. 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 [31], and the comparison of our approach with other variants of reinforcement learning methods.

Competing interests
The author declares that he has no competing interests.

Received: 8 October 2015 Accepted: 28 December 2015
Published online: 08 January 2016

References

1. L Xiang, J Luo, A Vasilakos, "Compressed data aggregation for energy efficient wireless sensor networks," in Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011 8th Annual IEEE Communications Society Conference on, june 2011, 46–54 (2011)
2. Y Song, L Liu, H Ma, AV Vasilakos, A biology-based algorithm to minimal exposure problem of wireless sensor networks. IEEE Trans. Netw. Serv. Manag. **11**(3), 417–430 (2014)
3. AV Vasilakos, GI Papadimitriou, A new approach to the design of reinforcement schemes for learning automata: stochastic estimator learning algorithm. Neurocomputing. **7**(3), 275–297 (1995)
4. L Liu, Y Song, H Zhang, H Ma, AV Vasilakos, Physarum optimization: a biology-inspired algorithm for the Steiner tree problem in networks. IEEE Trans. Comput. **64**(3), 819–832 (2015)
5. H Saad, A Mohamed, T ElBatt, Cooperative Q-learning techniques for distributed online power allocation in femtocell networks. Wirel. Commun. Mob. Comput (2014). doi:10.1002/wcm.2470
6. P Auer, NC Bianchi, Y Freund, RE Schapire, The nonstochastic multiarmed bandit problem. SIAM J. Comput. **32**, 48–77 (2003). doi:10.1137/S0097539701398375
7. K Shah, M Kumar, in *Proceedings of IEEE Mobile Adhoc and Sensor Systems*. Distributed Independent Reinforcement Learning (DIRL) Approach to Resource Management in Wireless Sensor Networks (IEEE, Pisa, Italy, 2007), pp. 1–9
8. MI Khan, B Rinner, in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops*. Resource Coordination in Wireless Sensor Networks by Cooperative Reinforcement Learning (IEEE, Lugano, Switzerland, 2012), pp. 895–900
9. M Khan, B Rinner, in *Proceedings of the IEEE International Conference on Communications Workshops*. Energy-Aware Task Scheduling in Wireless Sensor Networks Based on Cooperative Reinforcement Learning (IEEE, Sydney, Australia, 2014), pp. 871–877
10. C Frank, K Römer, in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*. Algorithms for Generic Role Assignments in Wireless Sensor Networks (IEEE, San Diego, California, 2005), pp. 230–242
11. JHW Ye, D Estrin, in *Proceedings of the INFOCOM'02*. An Energy-Efficient MAC Protocol for Wireless Sensor Networks (IEEE, New York, USA, 2002), pp. 1567–1576
12. Y Tian, E Ekici, F Ozguner, in *Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*. Energy-constrained Task Mapping and Scheduling in Wireless Sensor Networks (IEEE, Washington, DC, 2005), pp. 210–218
13. T He, S Krishnamurthy, JA Stankovic, T Abdelzaher, L Luo, R Stoleru, T Yan, L Gu, in *In Mobisys*. Energy-Efficient Surveillance System Using Wireless Sensor Networks (ACM Press, 2004), pp. 270–283
14. W Guo, N Xiong, H-C Chao, S Hussain, G Chen, Design and analysis of self adapted task scheduling strategies in WSN. Sensors J. **11**, 6533–6554 (2011). doi:10.3390/s110706533
15. X Xu, R Ansari, A Khokhar, AV Vasilakos, Hierarchical data aggregation using compressive sensing (HDACS) in WSNs. ACM Trans. Sens. Netw. **11**(3), 1–25 (2015)
16. S Giannecchini, M Caccamo, CS Shih, in *Proceedings of the Euromicro Conference on Real-Time Systems*. Collaborative Resource Allocation in Wireless Sensor Networks (IEEE, Rennes, France, 2004), pp. 35–44
17. T Meng, F Wu, Z Yang, G Chen, AV Vasilakos, Spatial reusability-aware routing in multi-hop wireless networks. IEEE Trans. Comput., 1–13 (2015). doi:10.1109/TC.2015.2417543
18. B Krishnamachari, S Wicker, R Bejar, C Fernandez, "On the complexity of distributed self-configuration in wireless networks". J. Telecommun. Syst. **22**(1–4), 33–59 (2003)
19. C Busch, R Kannan, AV Vasilakos, Approximating congestion + dilation in networks via "Quality of Routing" games. Int. J. Distrib. Wirel. Sens. Netw. **61**(9), 22 (2014)
20. S Dhanani, J Arseneau, A Weatherton, B Caswell, N Singh, S Patek, in *Proceedings of the IEEE Systems and Information Engineering Design Symposium*. A Comparison of Utility Based Information Management Policies in Sensor Networks (IEEE, Charlottesville, Virginia USA, 2006), pp. 84–89
21. P Li, S Guo, S Yu, AV Vasilakos, in *Proceedings of the IEEE INFOCOM*. CodePipe: An opportunistic feeding and routing protocol for reliable multicast with pipelined network coding, (Orlando, FL, 2012), pp. 100–108
22. RS Sutton, AG Barto, *Reinforcement Learning: An Introduction*. (MIT Press, Cambridge, Massachusetts, United States, 1998)
23. MI Khan, B Rinner, Performance analysis of resource aware task scheduling methodologies in wireless sensor networks. International Journal of Distributed Sensor Networks, Hindawi, Volume 2014, 11 (2014)
24. KLA Yau, P Komisarczuk, PD Teal, Reinforcement learning for context awareness and intelligence in wireless networks: review, new features and open issues. J. Netw. Comput. Appl. **35**, 253–267 (2012)
25. J Byers, G Nasser, in *Proceedings of the Workshop on Mobile and Ad Hoc Networking and Computing*. Utility Based Decision making in Wireless Sensor Networks (IEEE, Boston, MA, 2000), pp. 143–144
26. RAC Bianchi, CHC Ribeiro, AHR Costa, *Advances in Artificial Intelligence*. (Springer, Berlin, Germany, 2004)
27. DC Montgomery, EA Peck, GG Vining, *Introduction to Linear Regression Analysis*. (Wiley, Hoboken, New Jersey, United States, 2007), p. 152
28. N Bery, Linear regression. Technical report. DataGenetics (2009)
29. MR Spiegel, *Theory and Problems of Probability and Statistics*. (McGraw-Hill, New York City, New York, United States, 1992), pp. 116–117
30. T Abbes, S Mohamed, K Bouabdellah, Impact of model mobility in ad hoc routing protocols. Comput. Netw. Inf. Secur. **10**, 47–54 (2012)
31. L Esterle, PR Lewis, X Yao, B Rinner, Socio-economic vision graph generation and handover in distributed smart camera networks. ACM Trans. Sens. Netw. **10**(2), 24 (2014)