Energy-aware Task Scheduling in Wireless Sensor Networks based on Cooperative Reinforcement Learning

Muhidul Islam Khan, Bernhard Rinner
Institute of Networked and Embedded Systems
Alpen-Adria Universität Klagenfurt, Austria
Email: muhidulislam.khan@aau.at, bernhard.rinner@aau.at

***Abstract*—Wireless sensor networks (WSN) are an attractive platform for cyber physical systems. A typical WSN application is composed of different tasks which need to be scheduled on each sensor node. However, the severe energy limitations pose a particular challenge for developing WSN applications, and the scheduling of tasks has typically a strong influence on the achievable performance and energy consumption. In this paper we propose a method for scheduling the tasks using cooperative reinforcement learning (RL) where each node determines the next task based on the observed application behavior. In this RL framework we can trade the application performance and the required energy consumption by a weighted reward function and can therefore achieve different energy/performance results of the overall application. By exchanging data among neighboring nodes we can further improve this energy/performance trade-off. We evaluate our approach in an target tracking application. Our simulations show that cooperative approaches are superior to non-cooperative approaches for this kind of applications.**

***Index Terms*—Reinforcement learning, tasks scheduling, energy efficiency, wireless sensor networks, target tracking.**

I. Introduction

Wireless sensor networks (WSN) have become an attractive platform for various applications including target tracking, area monitoring or smart environments. Battery operated sensor nodes pose strong energy limitations where each sensor node has limited power supply, computation capacity and communication capability [1]. A typical WSN application is composed of different tasks which need to be scheduled on each sensor node. However, the scheduling of the individual tasks has typically a strong influence on the achievable performance and energy consumption.

The energy constrained sensor nodes operate in highly dynamic environments. Hence, the need for adaptive and autonomous task scheduling in wireless sensor networks is well recognized [2]. 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, the scheduler needs to consider the available energy of the sensor node as well as the energy requirements and the effect on the application's performance of each available task. The ultimate goal is to achieve a high application performance while keeping the energy consumption low.

In this paper we propose a cooperative reinforcement learning (RL) method for task scheduling. The proposed algorithm helps to learn the best task scheduling strategy based on the previously observed behavior and is further able to adapt to changes in the environment. A key step here is to exploit cooperation among neighboring nodes, i.e., the exchange of information about the current local view on the application's state. Such cooperation helps to improve the trade-off between energy consumption and performance. In our simulation we compare our cooperative with non-cooperative methods in terms of energy efficiency and application quality. We observe the energy/performance trade-off considering different balancing factors of the reward function, different network sizes and different target mobilities. The simulation results show that cooperative approaches are superior to non-cooperative or independent learning approaches.

The rest of this paper is organized as follows. Section II discusses related work, and Section III describes the problem formulation. Section IV explains our system model and the cooperative RL approach used. In Section V we present our RL based online task scheduling. Section VI discusses simulation results for an target tracking application. Section VII concludes this paper with a brief summary.

II. Related Works

In an energy constrained WSN, effective task scheduling is very important for facilitating the effective usage of energy [3]. 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 usage is optimized and also a considerable performance is maintained. Most of the existing methods of tasks scheduling do not provide online scheduling of tasks. They rather consider static task allocation instead of focusing on distributed task scheduling.

Guo et al. [4] proposed a self-adaptive task allocation/scheduling strategy in 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 trade-off among energy consumption and performance. Giannecchini et al. [5] proposed an online task scheduling mechanism called collaborative resource allocation (CoRAl) to allocate the network resources between the tasks of periodic applications in WSNs. CoRAl neither addresses mapping of tasks to sensor nodes

nor discusses explicitly energy consumption. Shah et al. [6] introduced a task scheduling approach for WSN based on an independent reinforcement learning algorithm for online tasks 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 trade-off. Our approach has some similarity with [6], but is much more general and flexible since we support general WSN topologies, a more complex reward function for expressing the trade-off between energy consumption and performance, and cooperation among neighbors.

III. Description of the Problem

In our approach the WSN is composed by 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 $ngh(n_i)$. The available energy of node n_i is modeled by a scalar E_i.

The WSN application is composed by 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 energy $\tilde{E}_j$ 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 (cp. Section V 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 fixed period. Thus, scheduling is required at the end of each period which is represented as time instant t_i. We only consider non-preemptive 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.

IV. 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 therefore apply reinforcement learning (RL) to determine the "best" task order given the experiences made so far. Figure 1 depicts our scheduling approach in terms of a RL framework where its key components can be described as follows.

Each sensor node represents an agent in our proposed multi-agent learning framework. The application represents the environment in our approach. 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 schedules the next task to execute. A state describes an internal representation of the application. State transitions depend on the previous state and action. The policy determines which task to execute at the present state. The policy can focus more on exploration or exploitation. It is built upon reward function values over time and hence it's quality totally depends on the reward function [6]. We apply a weighted reward function which is capable to show a trade-off between energy consumption and tracking performance. We consider the information exchange among neighbors which influences also the state of the application.

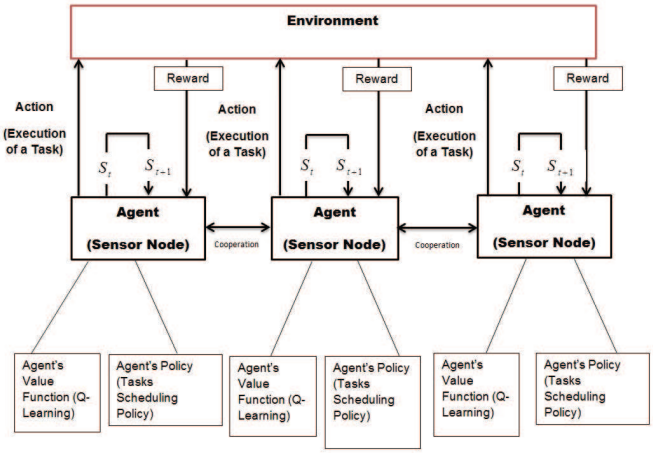

Fig. 1. Proposed system model.

Reinforcement learning is a branch of machine learning and is concerned with determining an optimal policy. It maps the states of the environment to the actions that an agent should take in those states so as to maximize a numerical reward over time [7].

Q learning [8] is a technique which is often used to select these actions, even when the agent has no full knowledge about the reward and state transition functions. In each state the agent basically can choose from two kinds of behavior: either it can explore the state space or it can exploit the information already present in the Q values.

SARSA(λ) [7], 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 such as a 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 updating rule for the Q values:

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha\delta_t e_t(s,a). \quad (1)$$

for all s,a.

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

$$\delta_t = r_{t+1} + \gamma f^i Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t). \quad (2)$$

In Equation 2, γ 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^i is the weight factor for the neighbors of agent i and can be defined as follows:

$$f^i = \frac{1}{ngh(n_i)} \qquad \text{if } ngh(n_i) \neq 0 \tag{3}$$

$$f^i = 1 \qquad \text{otherwise.} \tag{4}$$

An important aspect of an RL framework is the trade-off between exploration and exploitation [9]. Exploration deals with randomly selecting actions which may not have a higher utility in search of better rewarding actions, while exploitation aims at the learned utility to maximize the agent's reward.

SARSA(λ) improves learning through eligibility traces. $e_t(s,a)$ is the eligibility traces in Equation 1. Here λ is another learning parameter similar to α for guaranteed convergence.

The eligibility trace is updated by the following rule:

$$e_t(s,a) = \gamma\lambda e_{t-1}(s,a) + 1 \qquad \text{if } s = s_t \text{ and } a = a_t \tag{5}$$

$$e_t(s,a) = \gamma\lambda e_{t-1}(s,a) \qquad \text{otherwise.} \tag{6}$$

V. RL BASED TASK SCHEDULING FOR TARGET TRACKING

Tracking mobile targets is a typical and generic application for WSNs. We therefore demonstrate our task scheduling approach using such target tracking application. We consider a sensor network which may consists 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 time is energy demanding. Thus, 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. We propose a cooperative RL method for scheduling the tasks.

A. Set of Actions

We consider the following actions in our system:

a) Detect_Targets: This function scans the 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 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 origin and time (i.e., the current target position) 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, 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

$$v = \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2}/(t_t - t_{t-1}) \tag{7}$$

A slightly more advance estimation is based on the k least detected target positions, e.g., by exploiting regression or line fitting approaches.

e) Goto_Sleep: This function shuts down the sensor node for single time period. It consumes the least energy of all available actions.

f) 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

$$\tilde{t}_i = D_{P_i P_j}/v \tag{8}$$

where $D_{P_i P_j}$ is the distance between points, P_j and P_i correspond to the trajectory's intersection points with the FOV of the two nodes (cp. in Figure 2). v is the estimated velocity as calculated by Equation 7.

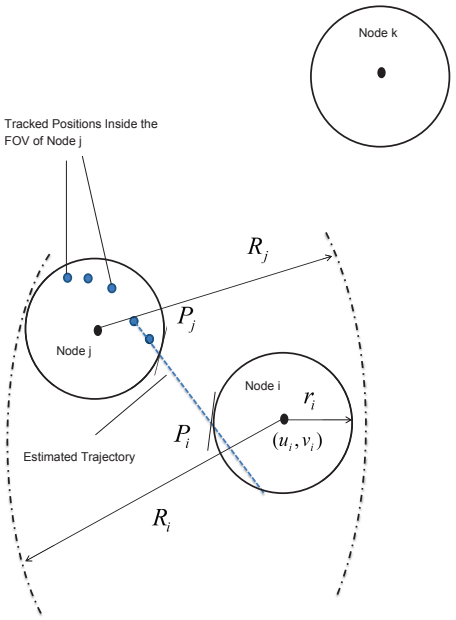

Fig. 2. Target prediction and intersection. Node j estimates the target trajectory and sends the trajectory information to neighbors. Node i checks whether the predicted trajectory intersects its FOV and computes the expected arrival time.

B. Set of States

We abstract the application 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 of any target already detected by some neighbor. If the time gap between local clock and the expected arrival time is greater than or

equal to five, the node remains in idle state. In this state, the sensor node performs $Detect\_Targets$ actions 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 five clock ticks. The threshold for the time difference between the expected arrival time and the local clock is set to five based on our simulation studies. In this state, sensor nodes perform $Detect\_Targets$ 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. This objective can be influenced by the balancing factor β of our reward function. 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 which 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. If the sensor node fails to execute the detection task then $N_t = 0$, i.e., there is no current detected targets inside 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. 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}.

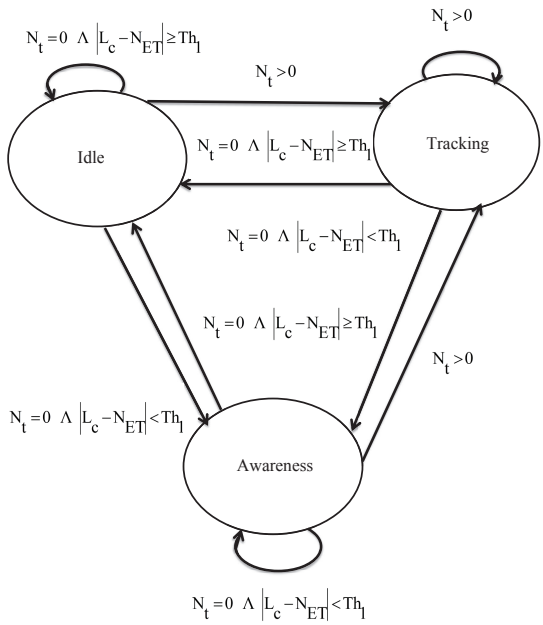

Fig. 3. State transition diagram. States change according the value of two application variables N_t and N_{ET}. L_c represents the local clock value and Th_1 is a time threshold.

C. Reward Function

The reward function in our algorithm is defined as

$$r = \beta(E_i/E_{max}) + (1-\beta)(P_t/P) \tag{9}$$

where parameter β balances the conflicting goals between E_i and P_t. E_i is the residual energy of the node. P_t is 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.

D. Exploration-Exploitation Policy

In our proposed algorithm, we use a simple heuristic where the exploration probability is represented by,

$$\epsilon = \min(\epsilon_{max}, \epsilon_{min} + k * (S_{max} - S)/S_{max}) \tag{10}$$

where ϵ_{max} and ϵ_{min} define 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).

Algorithm 1 SARSA(λ) 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 variable
    Select an action a, using policy
    Execute the selected action a
    Calculate reward for the executed action (Eq. 9)
    Update the learning rate (Eq. 11)
    Calculate the temporal difference error (Eq. 2)
    Update the eligibility traces (Eq. 5 and 6)
    Update the Q value (Eq. 1)
end while
```

Algorithm 1 shows the SARSA(λ) learning algorithm for the target tracking application step by step.

E. Learning Rate Update

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:

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

where ζ is a positive constant. $visited(s,a)$ represents the visited state-action pairs so far [10].

VI. Experimental Results and Evaluation

We evaluate our RL based task scheduling using a WSN multi-target tracking scenario implemented in a C# simulation environment. 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. Targets move around in the area based on a Gauss-Markov mobility model [11]. 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:

$$S_t = \eta S_{t-1} + (1-\eta)S + \sqrt{1-\eta^2}S^G_{t-1} \tag{12}$$

$$D_t = \eta D_{t-1} + (1-\eta)D + \sqrt{1-\eta^2}D^G_{t-1} \tag{13}$$

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^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:

$$x_t = x_{t-1} + S_{t-1}cos(D_{t-1}) \tag{14}$$

$$y_t = y_{t-1} + S_{t-1}sin(D_{t-1}) \tag{15}$$

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 I shows the energy consumption for the execution of each action. For each of our evaluations we run 10 simulations each lasting 100 time steps. We set the discounted factor $\gamma = 0.5$ for reinforcement learning and vary the learning rate according to Equation 11. We set $\zeta = 1$ for calculating learning rate in Equation 11. We set $k = 0.25$, $\epsilon_{min} = 0.1$, $\epsilon_{max} = 0.3$ and $S_{max} = 3$ in Equation 10. We set $\lambda = 0.5$ for the eligibility trace calculation by Equation 5 and 6. We consider the sensing radius, $r_i = 3$ and communication radius, $R_i = 8$. For each simulation run we aggregate the achieved tracking quality and energy consumption and normalize the tracking quality to $[0,1]$ and the energy consumption to $[0,10]$. As we get a value between 0 and 1 for calculating the tracking quality at every time steps, we normalize the tracking quality to $[0,1]$. Our highest amount of energy consumption for the Send_Message (two hops)=10 and the lowest amount is for Goto_Sleep=1. The send_message action requires the largest amount of energy. Sending messages over two hops consumes energy on both the sender and relay nodes. To simplify the energy consumption at the network level, we aggregate the energy consumption to 10 units on the sending node only. So, we normalize the energy consumption to $[0,10]$.

Action	*Energy Consumption*
Goto_Sleep	1 unit
Detect_Targets	2 units
Intersect_Trajectory	3 units
Predict_Trajectory	4 units
Send_Message (one hop)	5 units
Send_Message (two hops)	10 units
Track_Targets	6 units

TABLE I
ENERGY CONSUMPTION OF THE INDIVIDUAL ACTIONS.

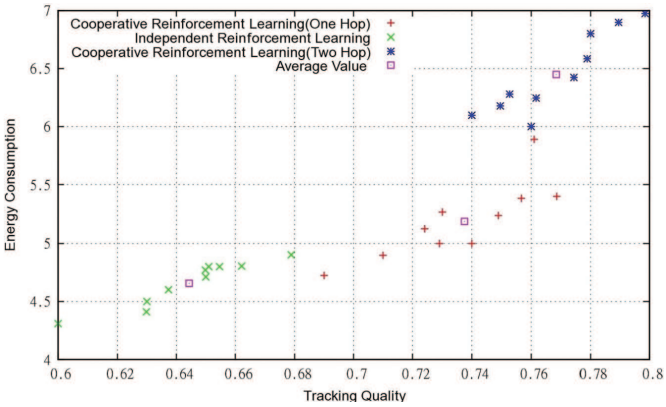

Fig. 4. Achieved trade-off between tracking quality and energy consumption for $\beta = 0.1$.

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

1) To find out the trade-off between tracking quality and energy consumption, we set the balancing factor β to one of the following values $\{0.10, 0.30, 0.50, 0.70, 0.90\}$,

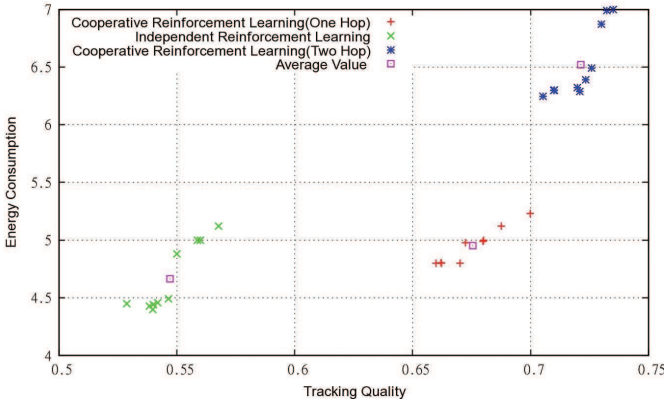

Fig. 5. Achieved trade-off between tracking quality and energy consumption for $\beta = 0.3$.

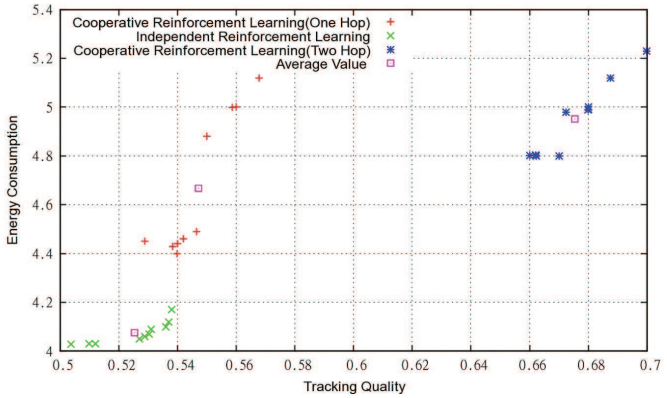

Fig. 6. Achieved trade-off between tracking quality and energy consumption for $\beta = 0.5$.

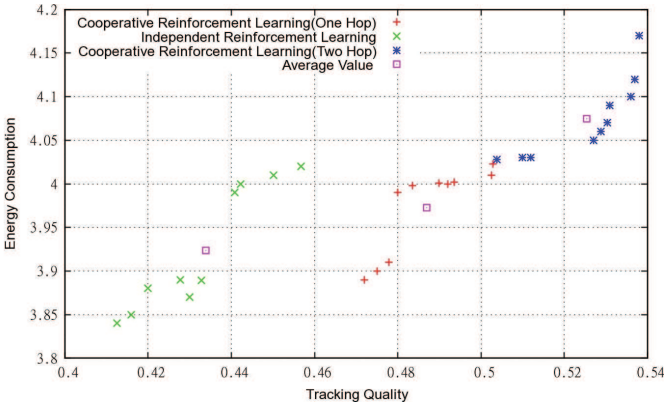

Fig. 8. Achieved trade-off between tracking quality and energy consumption for $\beta = 0.9$.

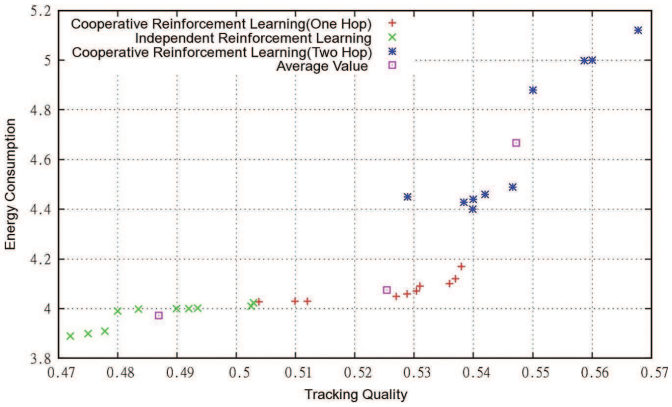

Fig. 7. Achieved trade-off between tracking quality and energy consumption for $\beta = 0.7$.

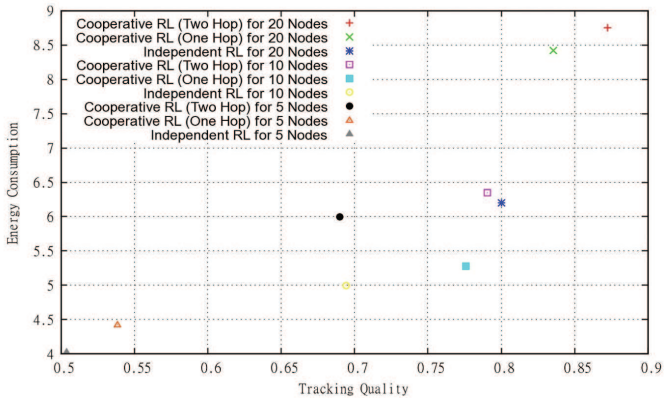

Fig. 9. Tracking quality versus energy consumption for various network sizes.

keep the randomness of moving target as $\eta = 0.5$ and fix the topology to five nodes.

2) We vary the network size to check the trade-off between tracking quality and energy consumption. We consider three different topologies consisting of 5, 10 and 20 sensor nodes. 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.

We compare our proposed cooperative approach (considering both one hop and two hop distance neighbors) with a non-cooperative or independent RL based task scheduling as reference for the above three experiments. Figures 4, 5, 6, 7 and 8 present the results of the first experiment. Each data point in these figures represents the normalized tracking quality and energy consumption of one complete simulation run. The square symbols represent the average values among the 10 simulation runs for each method. For example with $\beta = 0.1$, the achieved tracking results varies within $(0.69, 0.77)$ and the energy consumption varies within $(4.7, 5.4)$ for our one-hop cooperative approach. The average value for this setting is 0.73 and 5.3. It can be clearly seen from theses figures of the

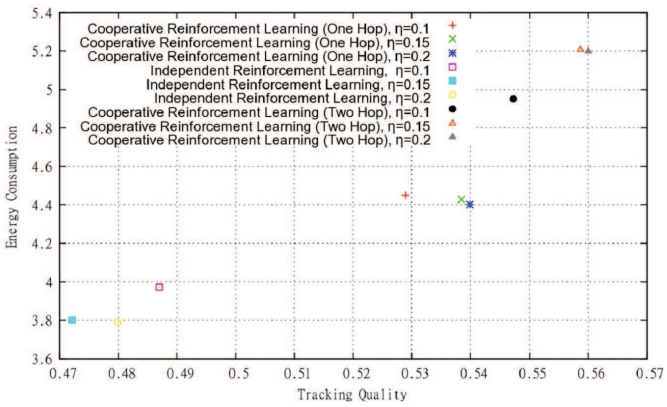

Fig. 10. Randomness of target movement, η=0.1, 0.15 and 0.2

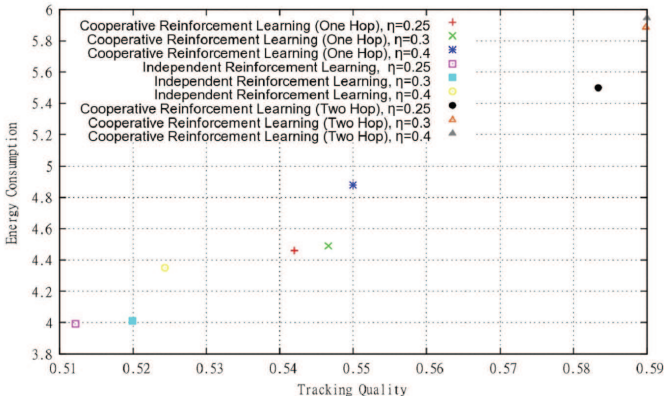

Fig. 11. Randomness of target movement, η=0.25, 0.3 and 0.4

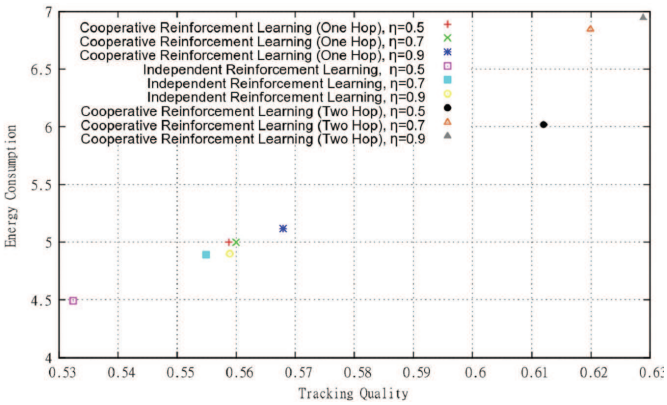

Fig. 12. Randomness of target movement, η=0.5, 0.7 and 0.9

first experiment that our cooperative approaches outperforms the non-cooperative approach with regard to the achieved tracking performance. There is a slight increase in the energy consumption especially for the two-hop cooperative approach. Figure 9 shows the results of our second experiment. Here the same trend can be identified as in the first experiment, i.e., the cooperative approaches outperform the non-cooperative approach with regard to the achieved tracking performance. Figures 10, 11 and 12 show the results of our third experiment. From these figures, it can be seen that our cooperative approaches outperforms the non cooperative approach in terms of achieved tracking performance. We can see that for lower randomness, η=0.5, 0.7 and 0.9, independent learning and one-hop cooperative learning show very close results for tracking performance. But for higher randomness, η=0.1, 0.15 and 0.2, independent learning gives poor performance with regard to tracking quality.

All three experiments demonstrate that cooperative RL based scheduling achieves better tracking performance than non-cooperative scheduling. Naturally, the cooperative approaches require more energy due to the increase communication effort. However, by appropriately setting the balancing factor β the desired performance or energy consumption can be achieved.

VII. Conclusion

Energy-aware effective tasks scheduling is very important for WSN to know the best task to execute on next time slots. In this paper, we proposed a cooperative reinforcement learning method for online scheduling of tasks in a way that the better energy/performance trade-off is achieved. We compared our proposed cooperative method (one hop and two hop distance neighbors) with non-cooperative methods. Our experimental results show that our cooperative RL based scheduling outperforms the non-cooperative scheduling in terms of tracking quality. Future works include the consideration of a real world motion model for the targets, the consideration of data association as a task and the comparison of our approach with other variants of reinforcement learning methods.

Acknowledgment

This work was supported 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, M. B. Wanja Hofer, Branislav Kusy, T. Schmid, Q. Wang, P. Dutta, and A. Terzis, "Low Power or High Performance? A Tradeoff Whose Time Has Come (and Nearly Gone)," in *Proceedings of European Conference on Wireless Sensor Networks*, 2012, pp. 98–114.

[2] 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*, 2012, pp. 895 – 900.

[3] C. Frank and K. Romer, "Algorithms for Generic Role Assignments in Wireless Sensor Networks," in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*, 2005.

[4] W. Guo, N. Xiong, H.-C. Chao, S. Hussain, and G. Chen, "Design and Analysis of Self Adapted Task Scheduling Strategies in WSN," *Sensors*, vol. 11, pp. 6533–6554, 2011.

[5] S. Giannecchini, M. Caccamo, and C. Shih, "Collaborative Resource Allocation in Wireless Sensor Networks," in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2004.

[6] K. Shah and M. Kumar, "Distributed Independent Reinforcement Learning (DIRL) Approach to Resource Management in Wireless Sensor Networks," in *Proceedings of IEEE Mobile Adhoc and Sensor Systems*, 2007.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[8] U. A. Khan and B. Rinner, "Dynamic Power Management for Portable, Multi-Camera Traffic Monitoring," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, 2012.

[9] 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*, 2000, pp. 143 – 144.

[10] U. A. Khan and B. Rinner, "Online Learning of Timeout Policies for Dynamic Power Management," *ACM Transactions on Embedded Computing Systems*, p. 25, 2013.

[11] 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.