Wireless Netw
DOI 10.1007/s11276-012-0496-2

Distributed resource management in wireless sensor networks using reinforcement learning

Kunal Shah · Mario Di Francesco · Mohan Kumar

Abstract In wireless sensor networks (WSNs), resource-constrained nodes are expected to operate in highly dynamic and often unattended environments. Hence, support for intelligent, autonomous, adaptive and distributed resource management is an essential ingredient of a middleware solution for developing scalable and dynamic WSN applications. In this article, we present a resource management framework based on a two-tier reinforcement learning scheme to enable autonomous self-learning and adaptive applications with inherent support for efficient resource management. Our design goal is to build a system with a bottom-up approach where each sensor node is responsible for its resource allocation and task selection. The first learning tier (micro-learning) allows individual sensor nodes to self-schedule their tasks by using only local information, thus enabling a timely adaptation. The second learning tier (macro-learning) governs the micro-learners by tuning their operating parameters so as to guide the system towards a global application-specific optimization goal (e.g., maximizing the network lifetime). The effectiveness of our framework is exemplified by means of a target tracking application built on top of it. Finally, the performance of our scheme is compared against other existing approaches by simulation. We show that our two-tier reinforcement learning scheme is significantly more efficient than traditional approaches to resource management while fulfilling the application requirements.

Keywords Wireless sensor networks · Resource management · Task scheduling · Reinforcement learning · Target tracking

K. Shah (✉) · M. Kumar
University of Texas at Arlington, Arlington, TX, USA
e-mail: kunal.shah@sabre.com

M. Kumar
e-mail: mkumar@uta.edu

M. Di Francesco
Aalto University, Espoo, Finland
e-mail: mario.di.francesco@aalto.fi

1 Introduction

Wireless sensor network (WSN) nodes are remarkably constrained in terms of their resources, i.e., energy, computational power and radio bandwidth. WSNs normally operate in uncertain and dynamic environments where the state of the system changes dynamically over time. In fact, new sensor nodes may join an already deployed network. On the other hand, existing nodes may cease to participate in WSN operations due to depleted batteries, malfunctions, or disconnection from the rest of the network. As a consequence, applications have to explicitly address the fact that WSNs are inherently dynamic and uncertain. At the same time, Quality of Service (QoS) requirements and optimization goals need to be fulfilled by the applications as well. Indeed, adaptive resource management is the key to any successful middleware solution to enable such applications.

Resource management includes initial sensor selection and task allocation as well as runtime adaptation of allocated tasks and resources. There are many proposed middleware solutions that advocate a strong need for proactive adaptation of resources [5, 6, 16, 31]. However, there are only few that have actually focused on adaptive resource management for WSN applications [6, 16]. The problem of resource management and adaptation (see Fig. 1) can be described as follows: *Given the application structure, the QoS requirements and the current system state, what is the* *best task-to-resource allocation strategy so that a given set of system-wide, application-driven, global parameters can be optimized?*

In the discussion above, the application structure is supposed to be defined in terms of the underlying tasks and their interactions. In this context, QoS requirements include metrics such as latency, reliability, coverage and so on. The current state of the system is defined by parameters which include, for instance, mobility, energy availability, and neighboring nodes. The optimization parameters include, but are not limited to, energy, bandwidth, and network lifetime. To be more specific, we illustrate the resource management problem by using a sample target tracking application. It can be defined in terms of the following tasks: *sample*, i.e., sense the environment for the presence of a target or for the occurrence of an event; *transmit* a message to next hop towards the a central collecting point (namely, a sink or base station); *receive* by turning on the radio and listening to incoming messages; *aggregate* two or more local and remote readings into a single reading (e.g., by performing data triangulation to better estimate the position of the target); *sleep* by setting the microcontroller and the radio in a low-power mode to minimize energy consumption. Here, the state representation may consist of the following variables: having one or more neighbors; being successful in recent sampling; being successful in recent message reception; and the quality of readings (e.g., the signal strength). Furthermore, QoS requirements here may include: tracking error; coverage area; and maximum allowed latency. Our goal in this case is to optimize energy usage among all sensor nodes. The goal of our resource management framework is to schedule and allocate tasks on each sensor node in the system, so that energy usage among all sensor nodes is minimized while fulfilling the requirements of the application. The target tracking application is used as a means to demonstrate the effectiveness of the proposed scheme, which can be easily applied also to diverse applications such as intrusion detection, data collection with mobile nodes, health monitoring, and so on.

In this article, we advocate the use of reinforcement learning to address the issue of dynamic resource adaptation in WSNs. We propose a bottom-up approach where each sensor node is responsible for task selection, as opposed to the top-down approach conventionally used by other middleware solutions. This bottom-up approach based on reinforcement learning allows the development of autonomous WSN applications with dynamic adaptation, minimal or no centralized processing for task allocation, and limited communication overhead. In order to make sure that the system is actually meeting the global application goals, we exploit a two-tier learning scheme. The first learning tier (micro-learning) allows individual sensor nodes to self-schedule their tasks by using only local information, thus enabling timely adaptation. The second learning tier (macro-learning) governs the micro-learners by tuning their operating parameters so as to guide the system towards a global application-specific optimization goal (e.g., maximizing the network lifetime). On one hand, micro-learners use Q-learning [28] as their independent reinforcement learning (RL) algorithm while, on the other hand, macro-learners use the Collective Intelligence (COIN) theory [9, 29, 30] to guide the system towards the global application-specific goal. We provide a detailed performance analysis of the proposed two-tier RL-based scheme, and compare it with other related approaches. We show that our scheme is very effective in optimizing the resource usage while fulfilling the application requirements.

The rest of the article is organized as follows. Section 2 overviews the related work available in the literature, while Sect. 3 introduces the background of the used techniques, i.e., Q-learning and the COIN theory. Section 4 details our Distributed Independent Reinforcement Learning (DIRL) approach consisting of micro-learners. Section 5 describes how concepts from the COIN theory can be utilized for proper guidance of micro-learners to ensure global optimality

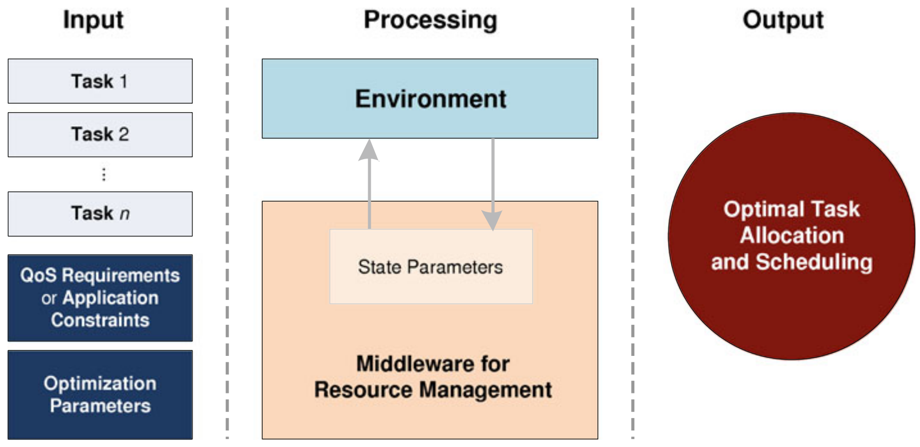

Input
Task 1
Task 2
⋮
Task n
QoS Requirements or Application Constraints
Optimization Parameters
Processing
Environment
State Parameters
Middleware for Resource Management
Output
Optimal Task Allocation and Scheduling

Fig. 1 The resource management problem in a WSN

and overcome the shortcomings of DIRL. Section 6 shows how our resource management framework can be applied to some real-world applications such as target tracking, data collection with mobile nodes, and health monitoring. Section 7 presents the simulation setup and Sect. 8 describes the results of the performance analysis of our two-tier RL-based framework. Finally, Sect. 9 concludes the article.

2 Related work

We present the related work available in the literature according to the different categories detailed below.

Rule and predicate logic Solutions falling in this category are based on a set of rules pre-defined on individual nodes. A rule fires if all conditions and parameters included in the rule predicate are satisfied, and this may result in task adaptation [4, 13, 25]. Even though being simple, this technique requires that all state conditions are known in advance, thus limiting the adaptation. Furthermore, this approach is unfeasible in very complex scenarios with a large number of nodes and highly varying system dynamics. Furthermore, the impact of rule-based adaptation at a local sensor node on the global WSN application is not considered. The Generic Role Assignment (GRA) scheme [4] considers nodes in a WSN system as taking a certain role in the network based on their properties, such that the requirements of a given role is satisfied. Impala [13] uses a set of rules to implement its own application adaptor component. The application adaptor in Impala follows a finite state machine where each state represents an application, and a transition between states is governed by rules represented as parametric expressions. FACTS [25] uses abstraction of facts, rules and functions. In the corresponding middleware architecture, every information—ranging from sensor readings to state variables—is represented as a fact. Facts, rules and functions are local to each node of the WSN which runs its own rule engine.

Constraint-satisfaction A different option is to define the problem in terms of constraint-satisfaction, which is sometimes reduced to linear programming with the objective function consisting of optimization parameters under given constraints (namely, the application requirements). The works in [10, 11, 17] are sample applications of the constraint-satisfaction approach to WSNs. Due to the complexity of the system, it is not always possible to reduce our resource adaptation problem into a linear programming problem without making unreasonable assumptions. In addition, in most cases the approach cannot be distributed, thus making it not practical for large-scale WSNs. A constraint-guided software reconfiguration approach is proposed in [10] where adaptation is performed by updating software components on TinyOS motes based on different constraints. In [11] various configuration tasks of multi-hop WSNs are mapped to distributed constraint-satisfaction problems. Specifically, the connection with critical transmission power threshold is derived, and it is shown that the average problem complexity can be reduced by tuning the transmission power of individual nodes. A distributed constraint-optimization algorithm called *Adopt* is proposed in [17] for generic multi-agent systems. Adopt uses localized asynchronous communication and makes local decisions based on conservative cost estimates rather than on global knowledge, and results in a polynomial-space algorithm.

Agent negotiation and auction theory Research in this category mainly includes multi-agent systems whose agents are able to negotiate with each other in order to determine the best resource allocation [12, 18, 26]. Specifically, the system consists of one or more mediators responsible for negotiations. Although this approach can lead to efficient resource management, the communication and computational resources required for negotiations may not be feasible in an implementation on resource-constrained wireless sensor nodes. A center-based algorithm called *Mediation* is proposed in [18] to address task assignment problem in distributed WSNs. A distributed resource allocation based on dynamic coalition formation and coalition strategy learning is presented in [12]. In this scheme, agents attempt to operate autonomously with incomplete information about their potential collaborators. Synchronization of actions across multiple agents is then achieved by forming coalitions via multiple 1-to-1 negotiations. However, because of the uncertainty in a WSN, the formed coalitions will be suboptimal. To allow adaptation to environment dynamics, each agent is capable of multiple levels of learning. This includes: case-based learning to learn about how to negotiate better; and reinforcement learning to learn how to form a better coalition. In our work, we also utilize multiple-levels of learning which are mainly based on reinforcements obtained from local actions and global outcome. We do not require any explicit coalition formation due to related significant communication and computation overhead, which is not feasible on resource-constrained sensor nodes.

Utility and market theory The purpose of the solutions in this category is to define a utility function for mapping an optimization parameter over a number of participating nodes to a real value. Then, such a function is maximized under given constraints, usually in terms of a linear programming problem, which can be addressed by either a distributed or centralized approaches. Utility functions to define global objectives of WSN applications along with cost model for energy consumption were first introduced in [2]. Here, a model where node makes heuristic assessments mostly based on locally available information is proposed. The model is further driven by objective functions that maximize the utility of WSNs over their lifetime. Although the simplistic model presented in that work is substantial, routing algorithms and heuristics presented are primitive and cannot be applied to real-world WSN applications. Our approach of allowing individual nodes to maximize their utility functions and the representation of cost model is actually inspired by [2]. A utility-based WSN system based on techniques from mechanism design and game theory is presented in [19]. The authors have attempted to address the same problem that we are interested here: designing local utility functions so that each node can selfishly optimize its local utility as well as to optimize of desired global objective function. However, the authors have concentrated on the theoretical aspects of developing a game-theoretic algorithm for constructing a load-balanced spanning tree in the network. Furthermore, the work in [19] does not consist in a generic framework for WSN management, which is the focus of this article. Self-Organizing Resource Allocation (SORA), a market-based approach for resource allocation in WSNs, is proposed in [15]. In SORA, each sensor node acts as an agent that tries to maximize its payment by undertaking a set of actions. Each action can result in some energy consumption and can also produce some goods with an associated price. Agents receive feedback on their actions in the form of payment which, in turn, characterizes their behavior.

Even though each of the approaches mentioned above can provide efficient resource management, they suffer from some pitfalls as described. None of these techniques tries to address uncertainty which is inherent in dynamic networks. Furthermore, most of them require a careful implementation of algorithms on a case-by-case basis, which may be quite difficult in WSNs. Therefore, a framework that can enable large set of applications with autonomous adaptation and minimum communication overhead is required.

3 Background

In this section, we will briefly describe Reinforcement Learning (Q-learning) and the concepts behind the COIN theory used in designing our resource management framework. Reinforcement learning (RL) is a branch of machine-learning and is concerned with determining an optimum policy that maps states of the world to the actions that an agent should take in those states so as to maximize a numerical reward signal [24]. Agent receives a numerical reward as a consequence of its action which provides a reinforcement signal. Agents try out different actions in order to learn what actions yield the most reward. Action is selected either based on past experiences (exploitation) or randomly (exploration). RL is very useful for interactive (online) learning in dynamic uncertain environments.

In this work, we use Q-learning [28] which is a form of model-free reinforcement learning. Q-learning uses a single data structure, an utility look-up table $Q(s, t)$ across states s and tasks t. The utility of performing task t in a state s is defined as the expected value of the sum of immediate reward r and the discounted utility of resulting state s after executing task t, i.e.,

$$Q(s,t) = E[r + \gamma e(s')|s,t]$$

where $e(s') = \max_t Q(s',t)$ over all possible tasks. Note that the expected value above is conditional upon being in state s and performing task t. As Q-learning is done online, the equation above cannot be applied directly as the stored utility values may not have converged yet to the final values. Hence, in practice, Q-learning is used with incremental step updates as given by the following expression:

$$Q(s,t) = (1-\alpha)Q(s,t) + \alpha(r + \gamma e(s'))$$

Here, $0 \leq \alpha \leq 1$ is the learning-rate parameter. It controls the rate at which an agent tries to learn by giving more (α close to 1) or less (α close to 0) weight to the previously learned utility value. Setting α equal to 1 will force the agent to ignore all previously learned utilities resulting in single-shot learning. $0 \leq \gamma \leq 1$ is a discount-factor: the higher the value, the greater the agent relies on future reward than the immediate reward.

An important aspect of an RL system is the trade-off between exploration and exploitation. Exploration deals with trying out some random actions which may not have higher utility in search of better rewarding actions, while exploitation tries to use the learned utility to maximize the agent's reward. Most of the RL system uses exploration with a certain probability ε, which can be a constant value (mostly around 0.1–0.5) or can be derived using some other heuristics like starting with a high value and gradually decreasing, for instance, by using the Boltzmann equation [24].

COIN [9, 29, 30] is a large multi-agent system (MAS) with a well-defined *world utility* function which rates the behavior of the entire system and there is little or no centralized control. Each agent in the MAS is *selfish* and runs a RL algorithm. Hence, the global behavior of the system is the collective effect of individual agents, each modifying their behavior by using a RL algorithm. COIN theory addresses the following design problem: *Ensure optimal world utility, given that each individual agent attempts to maximise its local utility.*

RL algorithms called *micro-learners* try to optimize the local utilities at each agent. The learning algorithms that update the utility functions of the agents are called *macro-learners*. COIN uses concepts from game-theory to devise a methodology for designing and updating the local utility functions at each agent so that system will approach the near-optimal values of the world utility. The COIN theory has established that a collective system which is factored and has higher learnability eventually reaches a Nash equilibrium point, where all nodes are fully rational in optimizing their utility functions. This Nash equilibrium point is also the Pareto optimal point of the system. We will address concepts used by COIN as well as its application to WSNs in Sect. 5.

4 Distributed independent reinforcement learning (DIRL)

The main idea of DIRL is to allow each individual sensor node to self-schedule its tasks and allocate local resources by learning the corresponding usefulness (utility) in any given state, while honoring application-defined constraints and maximizing total amount of reward over time [21]. The advantage of using independent learning is that no communication is required for coordination between sensor nodes since each node selfishly tries to maximize its own reward. The application can specify global optimization parameters in terms of rewards for individual tasks at sensor nodes. Thus, if an application needs to optimize energy usage, the reward function of each task can be expressed as a combination of the task output and the energy consumed in performing such task.

4.1 Assumptions and system model

Our model is based on the following assumptions.

1. Each node is able to perform only one task at a time, and all tasks are allocated according to *time steps* of duration τ.
2. Tasks are allocated to sensor nodes according to their features (in case of a heterogeneous network), and the initial task allocation is already done after the deployment.

Assumption 1 means that a sensor node is single-threaded, as it happens in many real-world sensor platforms available today. This is not a restriction, however, as our approach can be also used in environments supporting multi-threaded nodes. We are currently working on relaxing the constraint associated to Assumption 2 by designing a complete middleware framework with support for task distribution and task allocation.

In order to apply RL, we need to define the resource adaptation problem in terms of RL elements. These elements and their mapping are as follows.

- *Agent*: each sensor node corresponds to an agent in multi-agent reinforcement learning (MARL).
- *Environment*: the world surrounding the sensor node and its interactions.
- *Action*: the application task to schedule. An application is deployed on a sensor node in the form of a set of tasks that a node can perform, and each node schedules one task during each time step. For example, an agent may have the following set of actions: transmit, receive, sample, alarm, actuate, aggregate, etc.
- *State*: a set of application-defined and system variables. For example, one can include system variables such as the number of neighboring nodes, remaining energy, mobility, capability for outbound and inbound communications, and so on. Application-specific variables, such as sensor readings or signal strength, may also be part of the state. The number of states in a system can grow exponentially with the state variables, and most of the time it is not practical to enumerate all possible states in advance. To this end, DIRL uses a weighted Hamming distance to group similar states, thus reducing the total number of needed system states.
- *Policy*: determines what action an agent will take in a particular state. In our case, the policy determines which task to execute a certain sensor state. We have defined a policy that consists of predicates as well as an exploitation and exploration strategy. These as will be discussed later in this section.
- *Reward function*: provides a mapping of agent state and corresponding action to a reward (typically, a real number) that contributes to the utility. The goal of each agent is then to maximize total reward over time. In DIRL, this reward function needs to map the application-defined optimization parameters into a numerical reward. Each task in DIRL implements a simple reward function that determines the amount of reward (positive or negative) obtained during each execution of that task. For example, the reward of a receive task could be function of the received messages during its execution, as well as the amount of resources consumed. DIRL defines the reward as a function of expected price e_p, so that the reward can be tuned at runtime by simply updating the expected price e_p.
- *Value function*: defines the long-term objective of an agent as well as the possible future states not immediately as described by the reward function. Essentially, it is built upon the values of the reward function over time, hence its quality totally depends on the reward

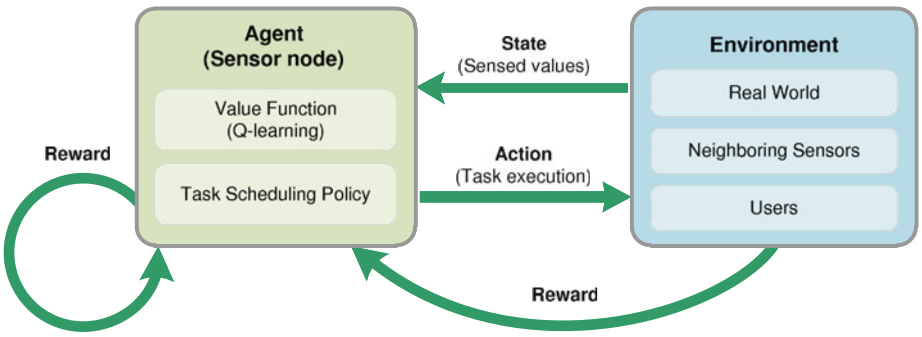


Fig. 2 Elements of reinforcement learning applied to a WSN

function. We use Q-learning, a form of RL that has an intrinsic value function defined.

Figure 2 illustrates the elements introduced above along with their mutual interactions.

4.2 DIRL framework

DIRL is based on Q-learning, a form of model-free reinforcement learning [28]. Q-learning is quite simple, demands minimal computational resources and does not require a model of the environment. Hence, it is very suitable to be implemented on resource-constrained sensor nodes. Furthermore, it also supports state-based learning by allowing sensor nodes to quickly adapt when a node switches from one state to another. Each sensor node in DIRL performs task and resource management by executing an algorithm based on Q-learning.

As outlined in Sect. 3, Q-learning uses a single data structure, namely, a utility look-up table $Q(s, t)$ that stores the knowledge acquired by the agent over time in the form of numerical utilities across states s and tasks t. For online Q-learning, the utility table is updated incrementally by using the following expression:

$$Q(s,t) = (1-\alpha)Q(s,t) + \alpha(r + \gamma e(s')) \tag{1}$$

Here, again, $0 \leq \alpha \leq 1$ is the learning-rate, while $0 \leq \gamma \leq 1$ is the discount-factor. In DIRL, we have used a simple heuristic where the exploration probability at any point of time is given by:

$$\varepsilon = \min(\varepsilon_{max}, \varepsilon_{min} + (S_{max} - S)/S_{max})$$

where: ε_{max} and ε_{min} define the upper and lower boundary for the exploration factor, respectively; S_{max} represents the maximum number of states (as obtained from the application) that DIRL will try to map; and S represents the current number of states already known. Thus the heuristic above allows an initial exploration with a higher rate, gradually decreasing over time as DIRL is able to discover (and map) more states. Note that some minimum exploration is always required to allow a node to dynamically reconfigure in case of environmental changes.

In DIRL, each node chooses a task to execute at each time step by either exploitation or exploration. However, all tasks may not be executable at all times. For example, the aggregate task cannot be executed if there are no readings available to aggregate. In addition, DIRL needs to honor certain application constraints, such as the latency and quality of readings, while scheduling tasks. In order to achieve this, DIRL associates each task with an applicability predicate that needs to be true for that task to be executed. Thus, a task is executed only if its applicability is satisfied. With reference to the *aggregate* task, if the application specified some constraints on the maximum latency of a reading, that can be reflected in the applicability predicate of the corresponding task. Figure 3 shows the flow diagram of task scheduling based on the exploration and exploitation policies, in addition to the applicability predicate of tasks at a particular time step τ.

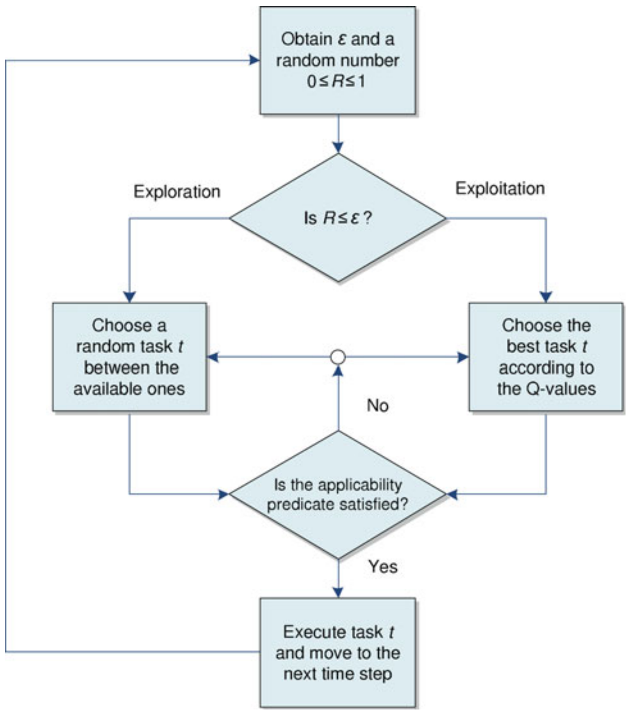

Obtain ε and a random number 0≤R≤1
Exploration
Is R≤ε?
Exploitation
Choose a random task t between the available ones
Choose the best task t according to the Q-values
No
Is the applicability predicate satisfied?
Yes
Execute task t and move to the next time step

Fig. 3 Task scheduling in DIRL

An important feature of a RL system is its handling of the temporal and structural credit assignment problem [14, 23]. The temporal credit assignment problem is to propagate the reward backwards in time, while the structural credit assignment is to propagate the reward spatially across states in order to define the notion of similarity. Q-learning provides support for the temporal credit assignment problem in terms of delayed reward [27]. The structural credit assignment also needs to be addressed, as otherwise each node will end up using a massive state-space, which is not practical for WSNs scenarios. DIRL uses a simple weighted Hamming distance between two states in order to address the structural credit assignment problem. While defining a state representation in the form of system and application variables, the application also specifies a weight associated to each variable. This weight is used by DIRL to define the state difference based on the variables. Thus, if the representation of an application state consists of variables $V_1, V_2, \ldots, V_n$ with the corresponding weights $W_1, W_2, \ldots, W_n$, DIRL determines whether two given states s_1 and s_2 are similar or not by calculating the corresponding Hamming distance as follows:

$$H(s_1 - s_2) = W_1 \cdot |(V_1(s_1) - V_1(s_2))| + W_2 \cdot |(V_2(s_1) - V_2(s_2))| + \ldots + W_n \cdot |(V_n(s_1) - V_n(s_2))| \qquad (2)$$

If the Hamming distance is less then a threshold, then two states s_1 and s_2 are considered to be similar and they share a single entry in the Q data-structure.

DIRL also needs the following inputs from the application:

- A set of application tasks in some priority order. Note here that the priority is important only until the Q-values are not established, or if two tasks have similar Q-values.
- An applicability predicate, associated to each task, incorporating application-specific constraints and reward functions towards the optimization goal.
- A state representation consisting of system and application variables, as well as the corresponding weight for determining Hamming distance and to aggregate similar states.
- The maximum number of states that DIRL should try to explore. This gives an upper bound on number of states in the system, so that DIRL will not try to identify any more states beyond this number. If the need arises, one can tune Hamming distance threshold to accommodate new states into the existing set of similar states.

Once the information above is available, DIRL performs the algorithm given in Fig. 4.

4.3 Global optimality

The main advantage of using independent learning in DIRL, as presented above, is that no communication is required for coordination between sensor nodes, and each node selfishly tries to maximize its own reward. This approach is feasible when each node in WSN application can acting on its own, and does not need to cooperate or compete with other nodes. In other words, if all nodes are working independently and their actions do not affect others, then any increase in the utility of individual nodes cannot reduce the utility of others and, hence, always results in increasing the world (system-wide) utility, which is merely the sum of all nodes utilities over all times. As we will see in the next section, such a system is subworld

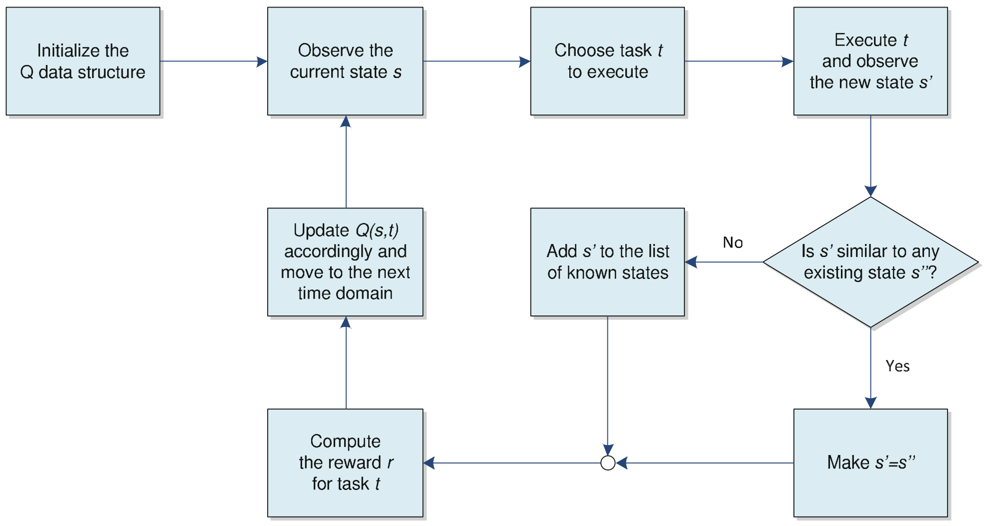

Initialize the Q data structure
Observe the current state s
Choose task t to execute
Execute t and observe the new state s'
Is s' similar to any existing state s''?
No
Yes
Add s' to the list of known states
Make s'=s''
Compute the reward r for task t
Update Q(s,t) accordingly and move to the next time domain

Fig. 4 The DIRL algorithm as performed by individual nodes

factored and will eventually reach a Pareto-optimal point, thus gearing towards the system-wide optimization goal.

However, most of the real-world WSN applications need some sort of cooperation among sensor nodes which cannot work independently. In this case, an increase in the utility of an individual node may result in the reduction of the utility of others and, hence, may not increase the world utility. It is also possible that such system is affected by the Tragedy of the Commons (TOC) phenomenon or Braes' Paradox [30], wherein the selfishness of individual agents leads to significantly lower global utility. Such phenomenon can be avoided by carefully designing the utility functions of the agents as well as the constraints for task selection. In other words, we need to make sure that the private utility is "aligned" with the global (world) utility, i.e., any increase in the private utility of an agent because of its own actions will also result in an increase of the world utility. A real-world example of this scenario is represented by human economy [29], wherein each individual tries to maximize their private utility in the form of income, career advancement, and so on. If there were no constraints on how individuals can operate and maximize their utility, people can harm others thereby leading to the downfall of the global utility (e.g., the gross domestic product). Here, government regulations act as the necessary constraints and modifications to utility function of individuals in order to ensure its alignment to global utility. Regulations are designed such that any increase in human utility will also cause an increase in the global growth of the economy.

In our framework, we neglect issues related to trust and reputation of sensor nodes, as the utility functions of all nodes in the system is under the control of the designer. As a consequence, all sensor nodes are assumed to be fair. Nevertheless, the problem of obtaining sub-optimal system-wide utility still needs to be addressed, even if nodes are not malicious. This is mainly related to imperfect (partial) knowledge that each node has about the rest of the system. This partial knowledge can lead a node to perform an action which may not benefit the overall system. Hence, the key in achieving a higher global utility lies in the design of private (to individual nodes) as well as global utility functions such that they are aligned.

Another drawback of using DIRL in real-world application is determining the reward functions and the price settings. It is difficult to investigate different aspects of system dynamics and choose the reward settings for each resource and task. Furthermore, these settings need to be changed on the fly whenever the overall system state and (or) the application requirements change. Hence, hand-tuning of rewards is not acceptable. Instead, COIN-based macro-learning can help learning the right settings for these rewards (private utilities) for individual nodes, and no domain expertise is required to set them. Macro-learning can update the utility function of micro-learners as and when the application states or requirements change, in order to move towards the global optimization goal.

In the next section, we will present further details about the COIN theory and how we have adapted it to the problem of resource management in sensor systems.

5 Collective intelligence-based macro learning

The problem of resource management in WSN can be defined in terms of COIN as follows [22]. Consider a WSN system, consisting of N nodes, evolving across a set of discrete, consecutive time steps $\tau \in \{1, 2, 3, \ldots\}$. Let $\xi_{n\tau}$ be an element of a vector space $Z_{n\tau}$, representing state of a node in our WSN at a particular time step τ. Here, state of a node consists of not only its application and system variables, but also node's actions that are directly visible to the outside world (including other nodes in the system). Following this convention, $\xi_\tau \in Z_\tau$ denotes a global state of our system, combining actions/variables from all nodes, at a particular time step τ, while $\xi \in Z$ is the vector of global state at all times. ξ can also be referred as world-line [29] in the space Z over all time steps. The goal of our framework is then to determine an optimal world-line ξ by maximizing some system-wide global utility function of ξ i.e., $G(\xi)$, and then driving the system along that world-line. Each node n in our framework is trying to maximize its private utility function say $q_n(\xi)$. Thus, we need to show here how and under what conditions our framework can determine the optimal ξ, given each node trying to maximize its private utility function $q_n(\xi)$, i.e., the Q-learning value function.

5.1 Application of COIN to WSN resource management

The major features of the COIN theory [9, 29] along with their applicability to the resource management problem in WSN are described below.

- A *collective* is a multi-agent system wherein each agent is adaptively trying to maximize its own private utility function, while, at the same time, there is a system-wide performance criterion defined to grade the behavior of the entire system. Thus, a WSN comprising individual sensor nodes adaptively trying to maximize their utility function according to a system-wide goal is a collective.
- Most of the collective system focus is on the *forward* problem of how local attributes induce a global behavior and thereby determine the system performance. On the other hand, COIN addresses the *inverse*

problem of designing a system to induce behavior that maximizes world utility. This is done by designing either private utility functions or incentives to private utility functions. Similarly, our objective is to design the reward and utility functions used by individual sensor nodes in the WSN so that the system-wide utility can be maximized.

- *Subworlds* are sets forming an exhaustive partition of agents. For each subworld w, all agents in that subworld share the same subworld utility function $g_w(\xi)$ as their local utility function. Accordingly, each subworld can be considered as a set of agents that collectively have the most significant effects on each other. In this case, agents cannot work against others, since all agents that affect each other substantially share the same local utility. As WSN applications are mostly data-centric, a chosen subworld is a set of sensor nodes involved in a data stream, i.e., from the data-source to the sink. For example, all nodes involved in a particular data stream from data sensors, to aggregators and collectors, will be part of a single subworld as they all have immediate and considerably high effect on each other. This subworld definition suggests that the subworld formation is dynamic and can change according to the state of the system. Figure 5 marks four sample data-stream subworlds in a target tracking application deployed over a 10-node WSN.
- A (perfectly) *constraint-aligned* system is such that any change in the state of agents in subworld w_i at time τ will have no effect on the states of agents in the subworld w_j $(i \neq j)$ at times greater than τ. Intuitively, a system is constraint-aligned if no two agents in separate subworlds affect each other, so that the rationale behind the use of subworlds holds. Most of the real-world systems are not perfectly constraint-aligned and the same holds in WSNs. In fact, a state change in a node involved in one data-stream (and hence one subworld) may affect state of other nodes in nearby data streams. But the effect here will be probably negligible if compared to the one on a node in the same data-stream.
- A *subworld-factored* system is such that, for each subworld w considered by itself, a change at time τ to the states of the agents in that subworld, when propagated across time, results in an increased value for $g_w(\xi)$ if and only if it results in an increase of $G(\xi)$. Mathematically, a system is subworld-factored if the following holds for all pair of states ξ and ξ' that differ only for subworld $w: g_w(\xi) \geq g_w(\xi') \iff G(\xi) \geq G(\xi')$. For a subworld-factored system, the side effects on the rest of the system of subworlds increasing their own utility do not end up decreasing world utility. In the current problem, if we model a system where each sensor node is a subworld, then it selfishly tries to maximize its private utility. This leads to a system that is not subworld-factored (assuming that sensor nodes are not totally independent) as the action of one node may be harmful to other more critical nodes and hence may reduce world utility. However, if we model a system where each data-stream is a subworld trying to maximize its utility by using an appropriate subworld utility function (e.g. Wonderful Life Utility Function [29]) $g_w(\xi)$ will not reduce the world utility because of the relative dependence with other data-streams during the existence of a subworld. Hence, such a system can be considered as subworld-factored.

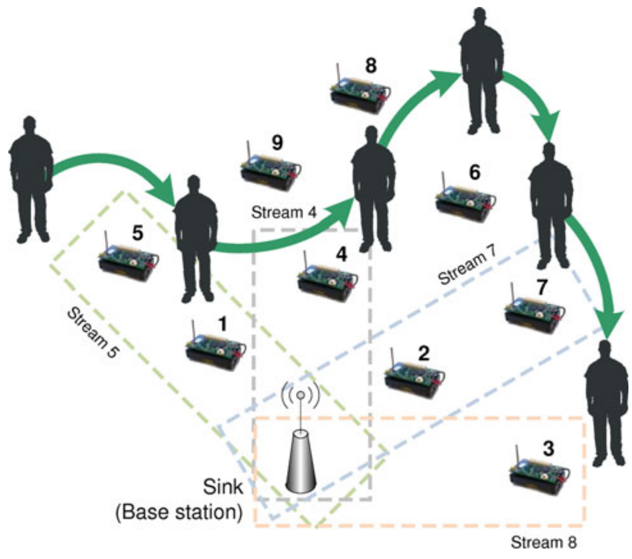


Fig. 5 Data-stream subworlds in a WSN application for target tracking

Another requirement for applying the COIN theory is to have private utility functions with high learnability. Learnability is a measure of how well an RL-algorithm can learn to optimize the utility function. For example, learnability of utility functions for a team game (where the reward of each node is the same as that for all nodes in the system at any time step) is much less than those of self-centered utility functions. Thus, the learnability of a utility function will be high if it is easy to interpret effect of node's action in the reward obtained. In DIRL, each node uses self-only utility functions where it gets immediate feed-back on the action taken and hence enjoys higher learnability. On the other hand, in COIN, all the nodes in a data-stream subworld share the same utility and, hence, the learnability of each node is lower than that of self-only utility functions, but considerably larger than those in a team game. This is because the number of nodes in single data-stream is just a fraction of the total number of nodes in a WSN. The wonderful life (WL) utility function plays an important role in designing a subworld-factored system because a constraint-aligned system with wonderful life subworld-utilities is subworld-factored. If $CL_w(\xi)$ is defined as vector ξ modified by clamping the states of all agents in subworld w across all time to a null vector (or say 0), then WL utility of w is:

$$g_w(\xi) = G(\xi) - G(CL_w(\xi)) \quad (3)$$

This definition of WL utility is the same as setting WL utility to world utility when considering that subworld w had never existed. Thus, the utility of a subworld will be high only if that subworld contribution has also increased world utility. We are using WL utility for macro-learning among sensor nodes which in turn is used to set the utility functions private to DIRL, i.e., the $q_n(\xi)$. The COIN theory proves that a collective system which is subworld-factored and has higher learnability, eventually reaches a Nash Equilibrium point where all nodes are fully rational in optimizing their utility functions [9, 29]. The COIN theory also shows that this Nash Equilibrium point is the Pareto optimal point of the system. From the above description, we can see that a resource management framework using a model of data-stream subworlds with Wonderful Life (WL) subworld utility function is subworld-factored and also has high learnability. Hence such a system will also eventually reach Nash Equilibrium which is also the Pareto optimal point and hence will avoid issues such as the Tragedy of Commons (TOC).

We will next describe how such a data-stream subworld scheme including macro-learning and settings of private utilities can be introduced to DIRL based resource management framework.

5.2 Resource management framework using two-tier learning

Our design goal is to create a system using a bottom-up approach where each sensor node is responsible for task selection, rather than the top-down approach (where some central entity dictates nodes what task to execute) used by many other middleware solutions [6, 16, 31]. The main advantages of bottom-up approach are: pro-active and real-time adaptation, no centralized processing requirement for task allocation, and minimal communication overhead. But principal challenge of bottom-up approach is how to make sure that system is actually meeting the global application goals and is not just acting randomly or creating chaos. We resolve this issue by using two-layer learning: micro-learning as used by individual nodes to self-schedule their tasks and macro-learning as used by each data-stream subworld to steer the system towards application goal by setting/updating rewards for micro-learners.

As mentioned earlier, the goal of a resource management framework is to determine the best allocation of tasks to sensors so that application-defined optimization goals—such as energy savings, network lifetime longevity, bandwidth preservation etc.—can be achieved while simultaneously honoring the QoS metrics of the application. QoS may be defined in terms of quality of measured variables (sensed and processed data) or other application constraints such as latency, reliability, and so on. Thus $G(\xi)$, which represents global utility over all time, can be expressed here as a sum of rewards $\sum_\tau R_\tau(\xi_\tau)$, where R_τ is global reward and ξ_τ is global state at time step τ. Thus, R_τ are temporal translations of one another, i.e.,

$$G(\xi) = \sum_\tau R_\tau(\xi_\tau) \quad (4)$$

As the global utility function may take many forms and is application-specific, we allow the application to define $R_\tau(\xi_\tau)$ given the current state of the system as represented by the measured variables (data) and optimization parameters. All the optimization parameters are represented in the form of a running-sum of numerical rewards, and are associated to each message.[1] It is also possible to provide a generic implementation of $R_\tau(\xi_\tau)$ based on the QoS requirements and the total of rewards from all data-streams. Each micro-learner uses Q-learning as in DIRL and, hence their private utility function $q_n(\xi)$ is a Q-learning value function as given by Eq. (1). On the other hand, macro-learners use the COIN-based wonderful life utility function. All sensor nodes that are part of one data-stream create a subworld and, hence will share same utility (reward) for single time step. From Eq. (3) and Eq. (4), the wonderful life reward of each agent part of subworld w at time step τ is given by:

$$g_{w\tau}(\xi_\tau) = R_\tau(\xi_\tau) - R(CL_w(\xi_\tau)) \quad (5)$$

In this case, $R(CL_w(\xi_\tau))$ is the world reward $R_\tau(\xi_\tau)$ after removing all data values that have been reported by data-stream w. This removes the effect of data-stream w on the world reward, but considers the actual contribution towards the world reward $R_\tau(\xi_\tau)$. Let us assume that data-stream w is providing values of a data variable which by itself is not significant, but has higher effect on the overall global application goal. In this case it is w which gets a higher reward as required. On the other hand, if w is contributing to redundant information provided by data-stream w' with higher reward, the wonderful life reward of w will be lower, as desired, thus discouraging its use. This reward value is used to update reward function of micro-learners for the task they executed for data-stream w. In order to

[1] In order to reduce the related overhead, we assume that the rewards are piggybacked into the messages.

perform resource management by using this COIN-based framework, we made the following extensions to the application input for DIRL described in Sect. 4.2.

- Instead of hand-tuning the expected prices associated to the reward function of each task (which we found very difficult to do given various system dynamics), the expected price is set by the macro-learner in terms of the WL utility of its subworld. Again, we have used a numerical price to allow a macro-learner to update private utility functions without incorporating new code.
- The application also provides a global reward function $R_\tau(\xi_\tau)$ which returns the global reward for a time step τ, given the current state of the system as represented by the measured variables (data) and the optimization parameters.

5.3 Architecture and system components

Each sensor node in the system embeds two agents: a micro-learner which is self-contained, trying to maximize its private utility by using only local information; and a macro-learner which is part of a subworld containing other sensor nodes linked in a data-stream and sharing the same utility functions. Once the application input is available, the system enters the *initialization phase*. This is needed since at the beginning there are no learned utilities available to either the micro-learners or the macro-learners. To this end, several options are possible.

- *Self-exploration and system learning*: RL-based systems always use a tradeoff between exploration (trying out random actions in search of better rewarding ones) and exploitation (choosing actions based on the obtained utilities) to build up its knowledge base. During the initialization phase, the rate of exploration needs to be higher compared to exploitation. Hence, decisions made by the system during this phase will be more random and may not immediately lead towards the system goal. This choice can be tolerated or not depending on the specific application and the WSN scenario under consideration.
- *Using domain knowledge*: domain expertise can be exploited to provide effective and faster initialization phase. This can be done in combination with self-learning for micro-learners. Domain knowledge can be provided to the system in the form of initial expected price (utility values set by macro-learners) for each task. This is similar to providing an initial estimate of the effect of tasks. These corresponding values can also be determined by using simulation and self-exploration as described above. As mentioned earlier, micro-learners have very high learnability and hence can build their utilities quickly.
- *Employing an available sensor-selection scheme*: any of the several sensor selection techniques (as mentioned in Sect. 2) such as MidFusion [1] or MiLAN [6]. Results from these techniques can be then used to initialize the macro-learners with the expected prices for individual sensor nodes.

After the initialization, micro-learners and macro-learners have some knowledge to exploit for decision making, and the system is then considered to be in the normal operating phase. During that time, the micro-learner tries to maximize its private utility function by running a modified version of the DIRL algorithm as given in Fig. 6. The micro-learner uses either exploration or exploitation for task selection at each time step τ based on the exploration

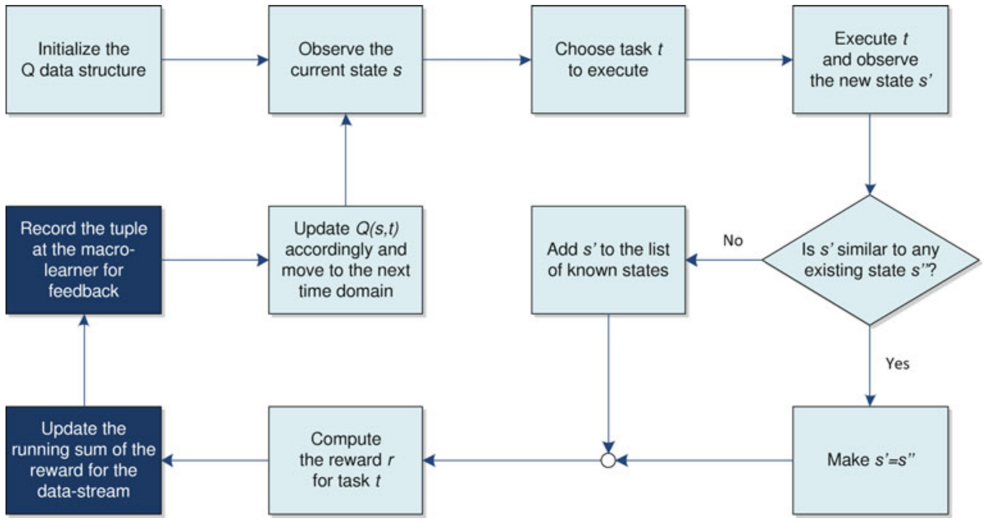

Initialize the Q data structure
Observe the current state s
Choose task t to execute
Execute t and observe the new state s′
Is s′ similar to any existing state s″?
No
Add s′ to the list of known states
Yes
Make s′=s″
Compute the reward r for task t
Update the running sum of the reward for the data-stream
Record the tuple at the macro-learner for feedback
Update Q(s,t) accordingly and move to the next time domain

Fig. 6 Micro-learning algorithm performed by individual nodes, where the blocks different from the plain DIRL have been marked with a *dark background*

factor, and also uses the Hamming distance between two states as the criterion to distinguish separate states, therefore reducing the state space of the sensor node. The micro-learner gets an immediate reward after task execution at each time step. The immediate reward is then used to update not only the Q-learning value function, but also the running sum of the reward ($r_{w\tau}$) on the message that it acted on. As the reward obtained is a function of application-specific optimization goal (e.g., minimizing the energy usage), $r_{w\tau}$ will be a measure of how well each data-stream is performing towards the application goal. The micro-learner also updates a running sum of cost ($c_{w\tau}$) as obtained from the cost function of a given task. A tuple consisting of the chosen task ID and the message ID at time step τ is recorded at the macro-learner so that it can provide future feedback on the related task execution.

The sink (base-station) is responsible for determining the wonderful-life reward $g_{w\tau}(\xi_\tau)$ (given by Eq. (5)) in a certain time step τ for each data-stream. Here, the global reward $R_\tau(\xi_\tau)$ is calculated by using the application-specific or generic global reward function, given the data stream output (measured variables), the total of running-sum of rewards, and the cost from all data-streams (as a measure of the optimization parameters). $CL_w(\xi_\tau)$ is also calculated by using the same global reward function, but discarding the reward obtained from w. The difference between the WL reward and the running sum of the reward based on the local utility of the data-stream given by δ_w, is next determined for each data-stream w through Eq. (6).

$$\delta_w = g_{w\tau}(\xi_\tau) - r_{w\tau} \tag{6}$$

The δ_w value is then handed over to the participating nodes in w along the reverse path of the data-stream. As each macro-learner gets its reward from the sink, it may arrive after a few time steps. As a consequence, the macro-learner maintains a history of the tuples recently recorded by the micro-learner (a tuple for each time step), until it receives a reward for that time step. The received reward is matched against the recent history by using the message ID, and is then used to update the utility value (i.e., the expected price of micro-learner) for the associated task. The expected price e_p of a task is updated by using the learning rate α as explained in Sect. 4.2, through the following equation:

$$e_p = (1-\alpha)e_p + \alpha\delta_w \tag{7}$$

The value of δ_w may need to be transmitted to each node participating in the data-stream w and may be a costly operation. To minimize this, we publish the WL reward only if it is significant as described below:

$$\begin{array}{ll} \text{Negative Reinforcement} & \text{if } -\delta_w < m < 0 \\ \text{Positive Reinforcement} & \text{if } 0 < m < \delta_w \\ \text{No Reinforcement} & \text{otherwise} \end{array} \tag{8}$$

where m is determined empirically based on the application-specific reward and the cost functions. The value of m can be varied to tune the effect of macro-learning on the system and as such a very high value of m may turn off macro-learning. If both $q_n(\xi)$ and $g_w(\xi)$ are correlated (i.e., they generate reward value using same metric), it is possible to design our global reward function so that $\delta_w \rightarrow 0$, as our system approaches a steady state. For example, if the value of $g_{w\tau}(\xi_\tau)$ is high for a data stream w, this will result in a high utility for the participating node $q_n(\xi)$. This will further increase the data-stream reward $r_{w\tau}$ in the subsequent data collection and thereby reduce the value of δ. As a consequence, the WL updates for the entire data-stream are required only when there is a state change resulting in a need for adaptation and for learning the new optimum task scheduling strategy. Updates will decrease and eventually be not needed any more as the system approaches a steady state.

The reinforcements for the macro-learners can be computed and transmitted by the sink according to a reinforcement window of N time steps, instead of at every time step τ. Therefore, the sink collects data for a window of N time steps and computes the reward of the stream based on the data accumulated in the window. This not only allows to save on communication costs, but also prevents the fluctuation of the global reward by computing it over a larger time interval.

Figure 7 gives a high-level overview of our framework and the related interactions with the application. As part of application deployment, the task graph and the associated application constraints—as well as the reward and cost functions—are disseminated to the nodes of the WSN. The application also provides a global reward function to the sink. The figure also shows the optional initialization of the local utilities of individual sensor nodes. The application can provide the variables of interest with the related QoS requirements at any state change. From this point onwards, each node takes the responsibility of self-scheduling its own tasks and allocating its own resources based on the local learned utilities. All data-streams are evaluated for WL reward at the end of each time step. The WL reward is next distributed to all the nodes in the corresponding data-stream, if significant, as determined by Eq. (8). Sensor nodes participating to the involved data-stream update their local-utility functions based on the global WL reward.

The set of micro-learners and macro-learners provides each sensor node with the capability to self-schedule tasks, while making sure that the overall system is guided towards its global optimization goal. Thus, such a scheme allows a sensor node to self-adapt to the system dynamics and the uncertainty inherent in the WSN. In fact, the

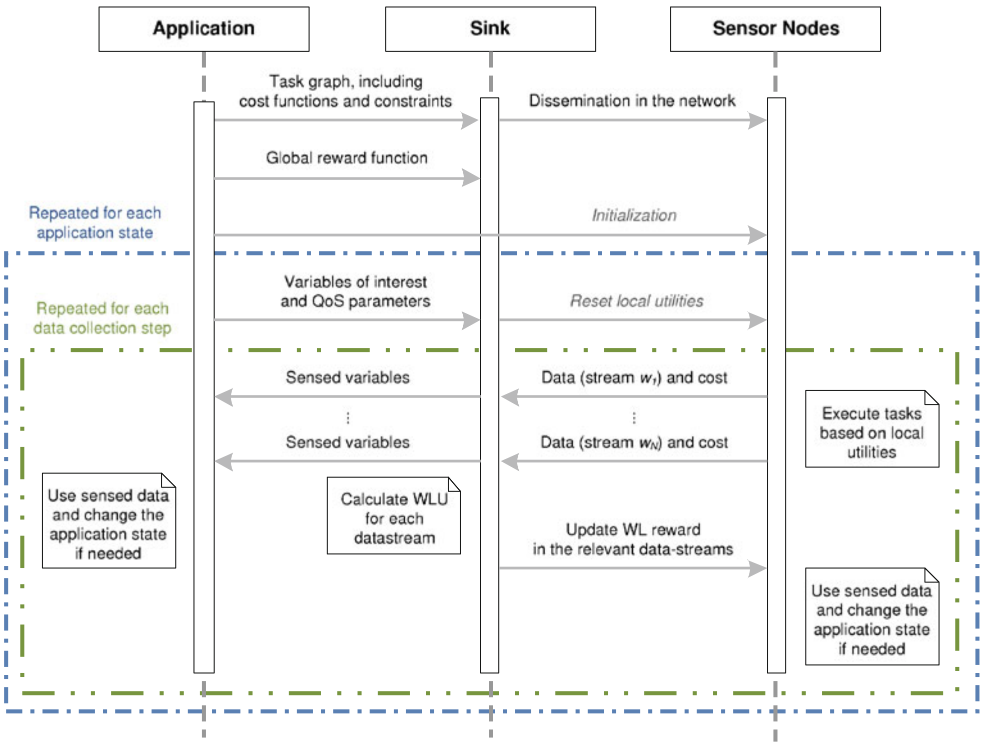

Application
Sink
Sensor Nodes
Task graph, including cost functions and constraints
Dissemination in the network
Global reward function
Repeated for each application state
Initialization
Variables of interest and QoS parameters
Reset local utilities
Repeated for each data collection step
Sensed variables
Data (stream w_1) and cost
Execute tasks based on local utilities
Sensed variables
Data (stream w_N) and cost
Use sensed data and change the application state if needed
Calculate WLU for each datastream
Update WL reward in the relevant data-streams
Use sensed data and change the application state if needed

Fig. 7 Summary of the interactions between the different components in our two-tier reinforcement learning scheme

micro-learner immediately adapts to changes in the local state (e.g., low battery, neighbor change, nearby target). In addition, the macro-learner provides adaptation at the global level with changes in the global state or the application requirements (e.g., change in QoS of data variables, addition and removal of sensor nodes). Global changes affect the WL utility of the macro-learner which, in turn, changes the utilities learned by the micro-learner along the corresponding direction. A change in the reward function of the micro-learners may invalidate the learned utilities, namely, the Q-values, which may not converge fast to the equilibrium [28]. However, it is possible to discard the learned utilities and start over again whenever the reward function is updated. In addition, here we are more interested in adaptation as well as convergence of the system as a whole rather than in the convergence of individual micro-learners.

6 Implementation of real-world applications

The design of our resource management framework is motivated by the need for flexibility, so as to allow different classes of WSN applications to be built on top of it. Applications which require autonomous adaptation in dynamic environments benefit the most from our framework. We will next show how some of real-world WSN applications can be easily implemented over our two-tier RL-based resource management framework. The variables of interest, QoS requirements, involved sensor types and global reward function for studied applications are given in Table 1.

6.1 Target tracking

A target tracking application can be deployed on top of heterogeneous WSN to track the presence of an object, a person, or even an event (e.g., fire propagation in the woods). As a consequence, this application can be used for intrusion detection, surveillance, or environmental and battlefield monitoring. Depending on the actual scenario, the application may have have different coverage area and lifetime requirements. In general, the application does not need redundant information provided by nodes with overlapping sensing ranges. In addition, tasks performed by each sensor node (e.g., sampling or routing) can be tuned based on current state of the system (e.g., presence of the tracked item or event). Hence, by means of efficient and continuous adaptive resource management over time, it is possible to allow sensors to preserve energy while still meeting application requirements.

Tasks involved in target tracking application along with the corresponding reward function and applicability predicate are defined below:

Table 1 Implementation of WSN applications

Element	Object tracking	Data collection with mobile elements	Health monitoring
Variables	Signal strength, covered area, location of the target	Presence of the mobile element at a given time, number of transferred messages	Heart rate, respiratory rate, blood pressure
QoS requirements	Signal strength greater than threshold and coverage and delay less than threshold	Detection accuracy, bandwidth utilization	Quality of measured variables greater than threshold
Types of sensors	Acoustic, proximity, video	Temperature, humidity, infra-red, chemical	ECG, blood pressure, blood flow, EMG
Global reward function ($R_\tau(\xi_\tau)$)	*f*(signal strength, coverage, tracking delay) − total cost of data acquisition	*f*(transferred messages) − total cost of discovery and communication	*f*(quality of measured variables) − total cost of data acquisition

- *Sample*. Obtains a sensor reading, i.e., the signal strength of a target object.
  Reward Function: (*noOfSensedEvents* $*$ *expectedPrice*) − *energySpent*
  Applicability Predicate: *remainingEnergy* $>$ *threshold*
- *Transmit*. Transmits a message to next hop towards the sink.
  Reward Function: (*noOfMsgsTransmitted* $*$ *expectedPrice*) − *energySpent*
  Applicability Predicate: *noOfOutboundMessages* > 0
- *Receive*. Sets the radio in receive mode to listen for incoming messages.
  Reward Function: (*noOfMsgsReceived* $*$ *expectedPrice*) − *energySpent*
  Applicability Predicate: always
- *Aggregate*. Aggregates two or more (local and/or remote) readings referring to the same target into single one (we use a simple *last value aggregation* in this example).
  Reward Function: (*noOfSamplesAggregated* $*$ *expectedPrice*) − *energySpent*
  Applicability Predicate: *noOfSamples* > 1 and *time FromLastReporting* $<$ *n steps*
- *Sleep*. Sets the microcontroller and the radio in sleep mode to minimize energy consumption.
  Reward Function: *expectedPrice* − *energySpent*
  Applicability Predicate: always

Table 2 shows the expected prices for the different tasks. Here, our goal is to optimize energy usage among all sensor nodes. This can be seen from the reward function which penalizes each task with the amount of energy consumed. The separation of the expected price from the reward function allows to dynamically tune the reward function by the macro-learner. Unfortunately, changing the reward function may invalidate the learned utilities (Q-values) which may not converge to the equilibrium (more details about the convergence of Q-values can be found in [28]). However, it is possible to discard the learned utilities and start over again whenever the reward function is updated. To this end, the values in Table 2 are only the initial estimates provided by system developer and the macro-learner will tune the values as required throughout the lifetime of the system. Also note that the expected price and reward functions are designed to have an effect only if the task execution succeeds. Thus if a node schedules the Receive task, then the node will obtain a positive reward only if one or more messages are received in the corresponding time step, otherwise it will receive a penalty proportional to energy consumed. The applicability predicates are quite simple and self-explanatory. Here, we do not allow a sensor node to sample if its energy is below a certain threshold, in order to make sure that the node is available for routing messages when required. This can be easily implemented in the applicability predicate for the Sample task as shown above.

Table 2 Expected price for tasks in the target tracking application

Name	Expected price
Aggregate	0.2
Transmit	0.1
Receive	0.2
Sample	0.05
Sleep	0.001

Our state representation consists of the following variables and their weights: have one or more neighbors (1.0), successful in recent sampling (1.0), successful in recent receive (1.0), signal strength (or quality of reading) (0.1). We used a Hamming distance of 1.0 as threshold, and maximum number of states was set to 5.

6.2 Data collection with mobile elements

Data collection can be performed in a WSN by means of mobile elements, i.e., either mobile sinks or special data nodes which relay collected data to the sink or base station [3]. In such a scenario, communication opportunities (namely, *contacts*) between a sensor node and a mobile element are limited. Furthermore, the contact duration is usually very short, especially if compared to the frequency in the arrivals of the mobile element. To this end, the goal of a data collection application should be to discover contacts between the sensor nodes and the mobile elements with the minimum energy consumption. At the same time, the contacts should be exploited as much as possible in terms of the successfully transferred messages.

A possible approach to the problem consists in defining an energy-efficient protocol for the timely discovery of the mobile element. At the sensors, the discovery protocol exploits a duty-cycle to save energy, such that individual nodes alternate active and sleep periods while trying to detect the proximity of a mobile element. Once the mobile element is discovered, the sensor node switches to a communication phase where it exchanges data with the mobile element until it is reachable so as to maximize the contact utilization. In this context, the application tasks can be defined as discovery tasks with different duty-cycles [20]. For instance, the following tasks can be defined.

- *High Rate Discovery*. The sensor nodes use a duty-cycle to the maximum allowed value Δ_{max}.
- *Low Rate Discovery*. The sensor nodes use a duty-cycle lower than Δ_{max}, but with the same order of magnitude.
- *Very Low Rate Discovery*. The sensor nodes use a duty-cycle significantly lower than Δ_{max}, i.e., one order of magnitude less.

The expected price associated to each task is proportional to the corresponding duty-cycle, and the state consists in the presence of the mobile element at a given time, which can eventually be further detailed as a function of the mobility model when available. Furthermore, the reward can be expressed as the successful discovery of the mobile element and the number of messages successfully transferred within a contact. Additional details for this scenario are provided in [20].

6.3 Health monitoring

In a health monitoring application, several sensors with different characteristics can be used to collect one or more physiological parameters and quantities related to activities. Each sensed value provides different quality of information, and is characterized by an associated cost. For an example, the heart rate can be measure by ECG, blood pressure monitor or blood flow monitor [6]. However, the accuracy and quality associated to the individual sensors are different, and so is the cost of obtaining the heart rate information. Intelligent resource management can help in choosing less costly sensors during normal conditions, and can trigger expensive but highly accurate sensor in case of emergency. The different aspects associated to such health monitoring application as required by our resource management framework are detailed in Table 1. The tasks for the health monitoring application are similar to those for target tracking, where the Sample task is responsible for measuring a given sensed variable (e.g., blood pressure or heart rate) and its reward function depends on the quality of the sensed variable.

7 Simulation setup

In order to evaluate our approach, we used the J-Sim simulator [8] and referred to the target tracking application, since it is the more demanding in terms of the learning process (see Sect. 6.1 for the details about the learning parameters).

We compared the performance of our two-tier learning approach (referred to as COIN) against the following schemes.

- *DIRL*: each node performs a task according to the DIRL approach. There is no macro-learning involved, and the system consists only of individual micro-learners as described in Sect. 4.
- *Random*: each node performs a task randomly chosen from a uniform distribution at each time step.
- *Simple*: each node performs a simple scheduling algorithm without trying to adapt or conserve energy by sleeping. As nodes are always active, this scheme provides the best tracking of the target object, as well as the upper bound on the energy usage.
- *Oracle*: this is an idealistic scheme that assumes each node somehow knows exactly what task to perform, and there is no overhead involved to manage the system. Thus, this scheme provides the lowest bound on energy usage, as well as the best tracking efficiency and accuracy. Clearly, this scheme cannot be implemented in practice and is used only for comparison purposes.

For our analysis, we considered the following performance metrics.

- *Global Reward*: this is related to the target events that are reported to the sink and the amount of energy consumed (i.e., the acquisition cost) as given in Table 1. As the optimization goal of our framework is defined in terms of the global reward function $G(\xi)$, the best way to measure its performance consists in using the global reward over time. The value of global reward is reported as percentage of value obtained from the Oracle scheme presented above. Thus, the global reward of Oracle scheme corresponds to 100 % and the other schemes have values relative to Oracle.
- *Activity Ratio*: this is defined as the ratio between the number of active tasks (Sample, TX and RX) and the total number of tasks executed in the system (hence, including the Sleep task). The activity ratio is averaged over all sensors in the system, and is expressed as a

percentage. A scheme with better resource management should have lower activity ratio as it should allow the system to conserve energy by executing the Sleep as much as possible, without affecting the accuracy significantly. A system where the nodes are always active (i.e., Simple) will have activity ratio of 100 %.

- *Energy consumption*: this is the average energy consumption spent by a single sensor node to track one object [7, 26]. In case of multiple tracked objects, the energy consumption is averaged over the different objects. Clearly, a lower energy consumption per tracked object indicates a better efficiency of the system.
- *Tracking Error*: this is the difference between the actual position of the tracked object and the position as estimated by the learning system. This metric is also averaged over all sensors and tracked objects in the system, and characterizes the tracking efficiency.

8 Simulation results

Simulation was performed under a variety of network and target scenarios which are detailed in the following sections. In all cases, sensor nodes are deployed in a grid of variable size, and their distance to the sink was up to two hops. Unless otherwise stated, the targets move along the grid with random direction and a constant speed of 3.6 km/h. In all experiments we performed 10 independent replicas, each of at least 10,000 s of simulated time. We also derived the confidence intervals with a 95 % confidence level. Table 3 summarizes the most important simulation parameters. The radio parameters were taken from the WSN package of J-Sim [8]. In the following, we will first focus on the learning performance, then we will investigate the impact of different number of tracked objects and sensor nodes on the considered metrics.

Table 3 Parameters used for simulation

Component	Parameter	Value
Micro-learner	Minimum exploration (ϵ_{min})	0.05
	Maximum exploration (ϵ_{max})	0.3
	Discount factor (γ)	0.5
	Learning rate (α)	0.5
	Time step (τ)	10 s
Macro-learner	Minimum WL reward (m)	0.25
	Reinforcement window (N)	10
Energy consumption	Sampling	84.1 μJ
	Routing	8.42 mJ
	Sleep	8.0 μJ

8.1 Learning performance

In this section, we evaluate our scheme in terms of its convergence to an equilibrium state, the reward of individual data-streams, as well as the global rewards. The results were obtained in a scenario with 10 sensor nodes and a single target randomly placed in a 300 × 300 m sensing area.

Figure 8 shows the convergence over time of our two-tier learning scheme COIN based on one simulation run (the corresponding behavior was found to be similar in multiple iterations as well) in terms of the reward of individual data-streams, i.e., δ_w as given by Eq. (6). In the considered scenario, the target was stationary and in range of multiple sensor nodes. For clarity, we also limited our analysis to a subset of meaningful data-streams as shown in in Fig. 5. We can see that initially multiple data-streams (i.e., 4, 5, 7 and 8) report a tracked object to the sink, and that δ_w oscillates between positive and negative values. This roughly corresponds to the initialization phase described in Sect. 5.2. After about 2,000 s, δ_w stabilizes: in this specific scenario, the system chose stream 4 for tracking the object, and the other (redundant) data-streams were turned off to preserve energy. At this stage, as $\delta_w \in (-m, m)$ for all streams, no more global reinforcement has to be sent out, thus the system has reached an equilibrium state. When the state of system or the application change, some variations in δ_w are again possible until the system reaches a new equilibrium. This behavior shows that our two-tier learning scheme is effective in selecting the best stream and in obtaining an equilibrium state rather quickly.

Figure 9 shows the global reward as a function of the simulation time for the different schemes. Different from the previous scenario, the target here was randomly moving along the grid with a speed of 3.6 km/h. We can see that COIN obtains the highest global reward (around 40 %), and is also able to maintain or even improve the reward over

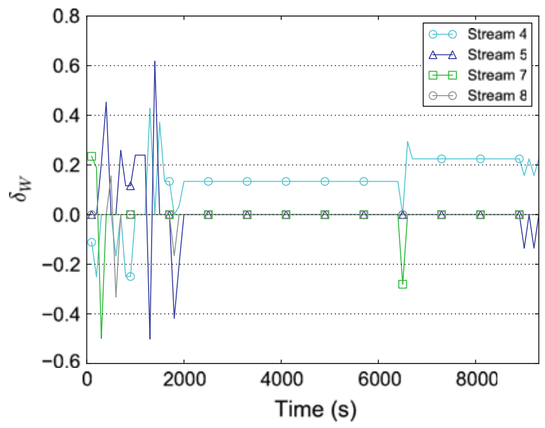


Fig. 8 Reward δ_w of individual data-streams as a function of time

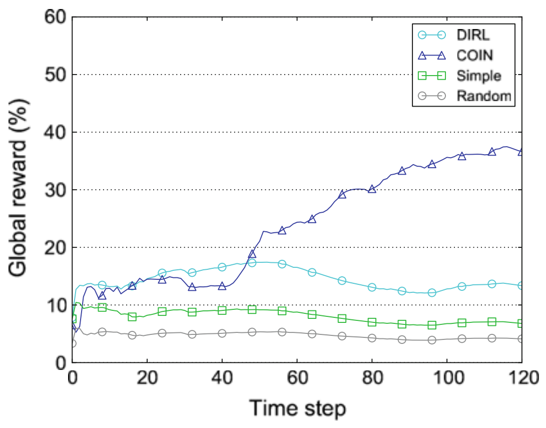


Fig. 9 Global reward as a function of time for the different schemes

time. This shows that COIN manages the system resources appropriately at all times by balancing the cost of acquisition and the reward from received data. As DIRL is based on micro-learning only, sensor nodes try to maximize their personal reward at all times and no consideration is given to system-wide performance. As a consequence, the system depletes considerable energy in redundant and unnecessary sensing and processing. As expected, Simple and Random have the lowest reward, as Simple keeps the nodes always on while Random does not actually exploit any learning.

8.2 Impact of the number of targets

In this scenario, 10 sensor nodes were deployed over a 300 × 300 m area, and the number of targets was varied from 1 to 3. All target objects were randomly placed over the grid, and moved along it with random direction with a speed of 3.6 km/h.

Figure 10(a) shows the global reward. We can see that COIN always outperforms the other schemes, even though the global reward decreases slightly with the number of tracked objects. Since the global reward is a function of the tracking events as well as amount of consumed resources, COIN manages the resources of the sensor nodes significantly more efficiently than the other schemes when there is one tracked object. When number of targets increases, more nodes are required to be active in order to successfully track all objects. Due to the higher number of targets, the system has a lower opportunity to conserve energy by resource management. As a result, the difference between COIN and the other schemes decreases when the number of targets increase. However, COIN still obtains a significant advantage over the other approaches even when the number of targets is equal to 3. Figure 10(b) illustrates the energy consumption. Here, COIN is closest to the (unfeasible) Oracle scheme independent of the number of targets. Specifically, the related energy consumption is almost a half of the one obtained by DIRL. Compared to the Simple scheme, which does not put the nodes to sleep, the energy consumption of COIN is always less than 25 %, even though it increases with the number of targets.

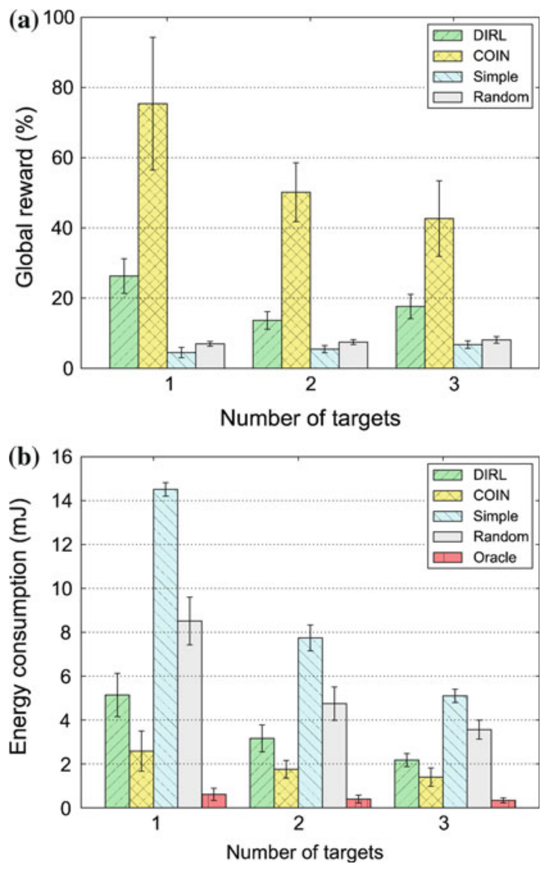


Fig. 10 Performance as a function of the number of targets: **a** global reward and **b** energy consumption

Figure 11(a) shows the activity ratio. We can observe that COIN maintains a low activity ratio of nearly 20 %, even when the number of targets increases. This shows the effectiveness of COIN in resource management. The activity ratio of DIRL increases with number of targets, while the one of Random is fixed and equal to about one third, i.e., the probability of randomly scheduling the sleep task out of available ones according to a uniform distribution. As nodes are always active, the Simple scheme always obtains an activity ratio of 100 %. Finally, Fig. 11(b) illustrates the tracking error. Clearly, the Simple scheme has the lowest tracking error since nodes cannot miss the target for being asleep. Indeed, sleeping enables energy conservation at the expense of the accuracy, namely, of the tracking error. Despite the lowest activity ratio, COIN still performs similar to DIRL. However, DIRL has a tracking error lower than COIN in general. This is related by the negative feedback received by datastreams which did not detect the target at a given time step k, but will eventually be in the sensing area of the target at

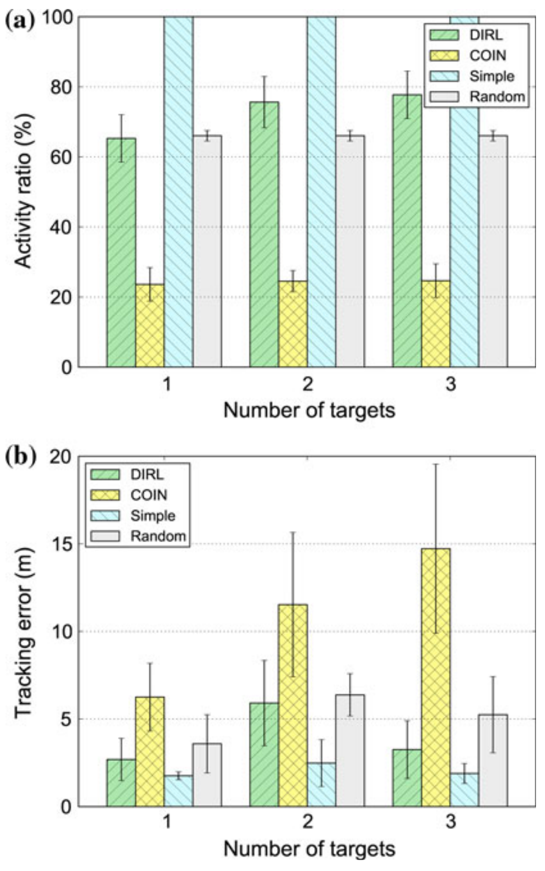


Fig. 11 Performance as a function of the number of targets: **a** activity ratio and **b** tracking error

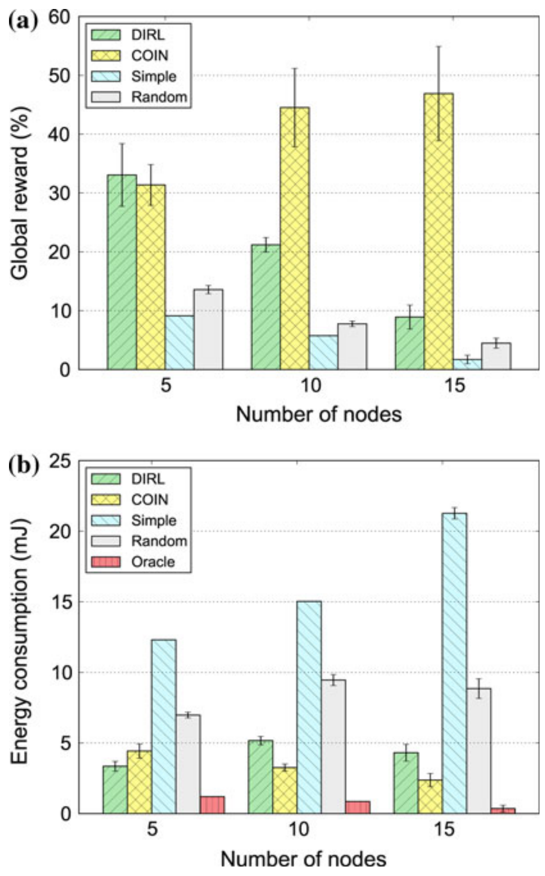


Fig. 12 Performance as a function of the number of nodes: **a** global reward and **b** energy consumption

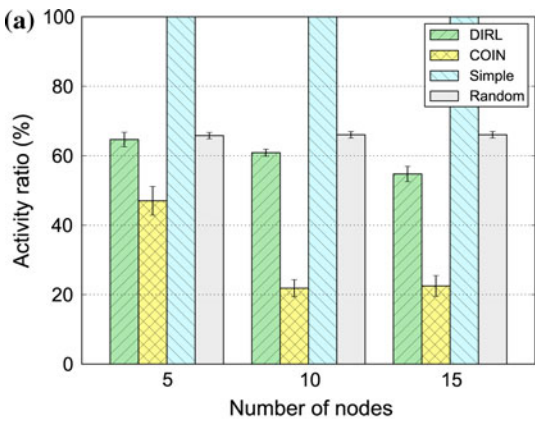
the $k + 1$th time step. This issue can be addressed by using a more sophisticated global reward function which uses the track of the target to predict its future direction and provide the reward accordingly. However, the solution presented here has the advantage of being able to track even events as they happen, and which may not follow an actual path.

8.3 Impact of the number of sensor nodes

In this scenario, we considered network deployments over a sensing area ranging from 200×200 m to 500×500 m, with different number of nodes (i.e., 5, 10, and 25), and with only a single target.

Figure 12(a) shows global reward. We can see that Simple and Random have the lowest reward, which also decreases when the number of sensor nodes increases. The figure also shows that the difference in the global reward between COIN and DIRL is almost the same when there are 5 sensor nodes in the network. This happens because it is more difficult to conserve energy when there are only a few sensor nodes, since most of them have to be relatively active to track the object. However, as the number of nodes increases, the difference in global reward between the different schemes also increases significantly. Specifically, the global reward decreases in all other schemes except for COIN. This shows that COIN is very effective in managing system resources as the redundancy and the overall size of the WSN increases. This is mainly due to the global feedback received by the sensor nodes participating in the associated data-stream. The efficiency of COIN is also apparent in Fig. 12(b), where the related energy consumption decreases when the number of nodes increases. Instead, the energy usage of the other schemes increases with the number of nodes, with Simple showing the maximum amount.

Figure 13(a) shows the activity ratio. Clearly, COIN has the lowest activity ratio, closely followed by DIRL, resulting in the lowest energy consumption. Similar to the previous figure, the overall activity ratio decreases when the number of nodes increases for COIN. Finally, Fig. 13(b) shows the tracking error. In this case, DIRL obtains a higher tracking efficiency than COIN, especially when the system changes are very dynamic (i.e., occurring during a few time steps). In fact, by the time global knowledge is learned and applied, the system may have moved to a different state, and this chance is higher when the motion of the target is fast and unpredictable.



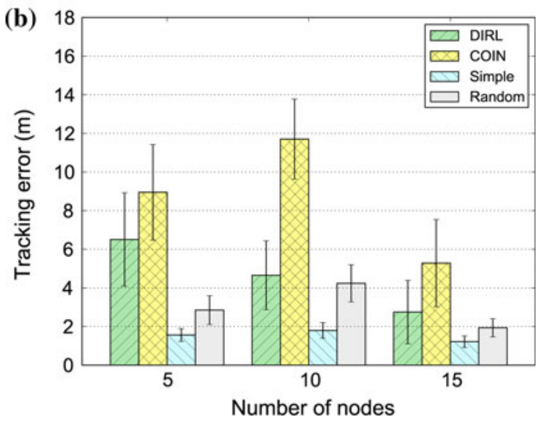


Fig. 13 Performance as a function of the number of nodes: **a** activity ratio and **b** tracking error

9 Conclusions

We presented a scheme for resource-management in wireless sensor networks (WSNs) that employs a bottom-up approach such that each sensor node is responsible for task selection. This approach is based on reinforcement learning and allows the development of autonomous WSN applications with dynamic adaptation, minimal or no centralized processing for task allocation, and low communication overhead. In order to ensure that system is actually meeting the global application goals, we used a two-tier learning scheme: micro-learning used by individual nodes to self-schedule their tasks; and macro-learning used by each data-stream to guide the system towards application-defined goals. Specifically, we used the collective intelligence (COIN) theory to enable macro-learning by setting and updating the operating parameters of the micro-learners. Simulation results showed that two-tier learning can substantially improve the overall performance compared to micro-learning or macro-learning alone. The application of the COIN theory guarantees that the Pareto-optimal point is eventually achieved, and avoids the system to be confined in a local maximum.

As a future work, we are looking into refining the interactions between micro-learners and macro-learners as well as the reward distribution, in order to enable private utilities under more complex application scenarios which are not limited to data-streams. A complete middleware framework is under development to support the proposed resource management scheme including the associated task interactions and node coordinations. Furthermore, we will investigate health monitoring applications as they will benefit from the two-tier approach.

Acknowledgments The research work presented in this article was supported by the National Science Foundation (NSF) under Grants ECCS-0824120 and CNS-0721951.

References

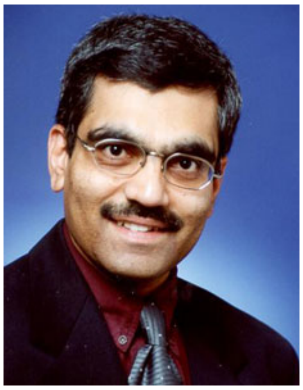
1. Alex, H., Kumar, M., & Shirazi, B. (2008). Midfusion: An adaptive middleware for information fusion in sensor network applications. *Information Fusion, 9*(3), 332–343. doi:10.1016/j.inffus.2005.05.007.
2. Byers, J. W., & Nasser, G. (2000). Utility-based decision-making in wireless sensor networks. In: *MobiHoc* (pp. 143–144). ACM. doi:10.1145/514151.514178.
3. Di Francesco, M., Das, S. K., & Anastasi, G. (2011). Data collection in wireless sensor networks with mobile elements: A survey. *ACM Transactions on Sensor Networks 8*(1). doi:10.1145/1993042.1993049.
4. Frank, C., & Römer, K. (2005). Algorithms for generic role assignment in wireless sensor networks. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.8091; http://www.vs.inf.ethz.ch/publ/papers/sensys05.roleassignment.ps.
5. Hadim, S., & Mohamed, N. (2006). Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online 7*(3), Article no. 0603–o3001.
6. Heinzelman, W., Murphy, A., Carvalho, H., & Perillo, M. (2004). Middleware to support sensor network applications. *IEEE Network 18*,(1), 6–14.
7. Intanagonwiwat, C., Govindan, R., & Estrin, D. (2000). Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MOBICOM* (pp. 56–67). doi:10.1145/345910.345920.
8. J-sim: Component-based, compositional simulation environment. http://sites.google.com/site/jsimofficial/.
9. Kagan Tumer, D. H. W. (2004). *Collectives and the design of complex systems*. Berlin: Springer.
10. Kogekar, S., Neema, S., Eames, B., Koutsoukos, X., Ledeczi, A., & Maroti, M. (2004). Constraint-guided dynamic reconfiguration in sensor networks. In: *Proceedings of the 3rd international symposium on information processing in sensor networks (IPSN-04)* (pp. 379–387). New York: ACM Press.
11. Krishnamachari, B., Wicker, S. B., Béjar, R., & Fernández, C. (2003) On the complexity of distributed self-configuration in wireless networks. *Telecommunication Systems 22*(1–4), 33–59. doi:10.1023/A:1023426501170.
12. Leen-Kiat Soh Costa Tsatsoulis, H. S. (2003). Distributed sensor networks: A multiagent perspective, chap. In *A satisfying, negotiated and learning coalition formation architecture* (pp. 109–137). Kluwer.

13. Liu, T., & Martonosi, M. (2003). Impala: A middleware system for managing autonomic, parallel sensor systems. In: *PPoPP'03: Proceedings of the 9th ACM SIGPLAN symposium on principles and practice of parallel programming* (pp. 107–118).
14. Mahadevan, S., & Conell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence 55*(2), 311–365.
15. Mainland, G., Parkes, D. C., & Welsh, M. (2005). Decentralized, adaptive resource allocation for sensor networks. In *NSDI'05: Proceedings of the 2nd symposium on networked systems design & implementation* (pp. 315–328).
16. Marron, P. J., Lachenmann, A., Minder, D., Hahner, J., Sauter, R., & Rothermel, K. (2005). TinyCubus: A flexible and adaptive framework sensor networks. In: *EWSN'05: Proceedings of the 2nd European workshop on wireless sensor networks* (pp. 278–289).
17. Modi, P. J., Shen, W., Tambe, M., & Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence, 161*(1–2), 149–180.
18. Ortiz, C., Rauenbush, T., Hsu, E., & Vincent, R. (2003). Distributed sensor networks: A multiagent perspective. In *Dynamic resource-bounded negotiation in non-additive domains* (pp. 61–106). Kluwer.
19. Sadagopan, N., Singh, M., & Krishnamachari, B. (2006). Decentralized utility-based sensor network design. *MONET 11*(3), 341–350. doi:10.1007/s11036-006-5187-8.
20. Shah, K., Di Francesco, M., Anastasi, G., & Kumar, M. (2011). A framework for resource-aware data accumulation in sparse wireless sensor networks. *Computer Communications 34*(17), 2094–2103. doi:10.1016/j.comcom.2011.06.010.
21. Shah, K., & Kumar, M. (2007). Distributed independent reinforcement learning (DIRL) approach to resource management in wireless sensor networks. In: *IEEE internatonal conference on mobile adhoc and sensor systems, 2007 (MASS 2007)* (pp. 1–9). doi:10.1109/MOBHOC.2007.4428658.
22. Shah, K., & Kumar, M. (2008). Resource management in wireless sensor networks using collective intelligence. In: *International conference on intelligent sensors, sensor networks and information processing, 2008 (ISSNIP 2008)* (pp. 423–428). doi:10.1109/ISSNIP.2008.4762025.
23. Sutton, R. (1984). *Temporal credit assignment in reinforcement learning*. Ph.D. thesis, Department of Computer Science, University of Massachusetts, Amherst, MA. Published as COINS Technical Report 84-2.
24. Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: The MIT Press. http://www.cs.ualberta.ca/sutton/book/ebook/the-book.html.
25. Terfloth, K., Wittenburg, G., & Schiller, J. H. (2006). FACTS—a rule-based middleware architecture for wireless sensor networks. In *COMSWARE IEEE*.
26. Victor Lesser Charles Ortiz, M. T. (2003). *Distributed sensor networks: A multiagent perspective*. Kluwer.
27. Watkins, C. (1989). *Learning from delayed rewards*. Ph.D. thesis, University of Cambridge, England.
28. Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning 8*(3), 279–292.
29. Wolpert, D., & Tumer, K. (1999). An introduction to collective intelligence. CoRR **cs.LG/9908014**. http://arxiv.org/abs/cs.LG/9908014 (Informal publication).
30. Wolpert, D., & Tumer, K. (2002). Collective intelligence, data routing and braess' paradox. *Journal of Artificial Intelligence Research (JAIR), 16*, 359–387. doi:10.1613/jair.995.
31. Yu, Y., Krishnamachari, B., & Prasanna, V. K. (2004). Issues in designing middleware for wireless sensor networks. *IEEE Network, 18*(1), 15–21.

Author Biographies

Kunal Shah's research interests include design and development of adaptive middlewares for wireless sensor and pervasive networks, application of reinforcement learning, utility theory and distributed computing. Kunal received his Bachelor of Engineering degree from SVNIT Surat, India in 1999. He started his career working as a Lecturer at Nirma University. After finishing his Masters in Computer Science in 2003, Kunal worked full time as Design Software Engineer for SensorLogic Inc. for 5 years. He began his Ph.D. in 2005 at the University of Texas at Arlington. Since 2011, he is working as a Team Lead at Sabre Holdings.

Mario Di Francesco is a post-doctoral researcher at the Department of Computer Science and Engineering of Aalto University, Finland. He is also an adjunct faculty member at the Department of Computer Science and Engineering of the University of Texas at Arlington, USA. His research interests include wireless sensor networks, pervasive computing, mobile networking, and performance evaluation. He has been in the technical program committee of the International Workshop on Sensor Networks and Systems for Pervasive Computing (PerSeNS) since 2009. Mario obtained his PhD degree from the Department of Information Engineering of the University of Pisa in May 2009, under the supervision of Prof. Giuseppe Anastasi. He was a research fellow at the St. Anna's School of Advanced Studies in 2009.



Mohan Kumar's current research interests are in pervasive computing, wireless networks and mobility, sensor systems, information processing and distributed computing. He has published over 140 articles in refereed journals and conference proceedings and supervised several doctoral dissertations and Masters theses in the above areas. He is a co-founder of the IEEE International Conference on pervasive computing and communications (PerCom). Kumar is one of the founding editors of the Pervasive and Mobile Computing Journal and is on the editorial board of The Computer Journal. Previously he held faculty positions at the Curtin University of Technology, Perth, Australia (1992–2000), The Indian Institute of Science (1986–1992), and Bangalore University (1985–1986). Dr. Kumar obtained his Ph.D. (1992) and M.Tech. (1985) degrees from the Indian Institute of Science and the B.E. (1982) from Bangalore University in India. He is a senior member of the IEEE.