Distributed Resource Management in Wireless Sensor Networks using Reinforcement Learning

Kunal shah

and

Mohan kumar

The University of Texas at Arlington, Arlington TX 76019

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*In wireless sensor networks, 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 development of scalable and dynamic wireless sensor network applications. We present two-tier reinforcement learning based resource management framework to enable autonomous self-learning/adaptive applications with inherent support for efficient resource management.* *Our design goal is to create a system using a bottom-up approach where each sensor node is responsible for its resource allocation/task selection. First tier learning (micro-learning) enables individual sensor nodes to self-schedule their tasks using only local information allowing for a real-time adaptation. Second tier learning (macro-learning) governs the micro-learners by setting their utility functions in order to steer the system towards application’s global optimization goal (e.g. maximize network lifetime etc). We exemplify the effectiveness of our framework by designing a tracking/surveillance application on top of it. Finally, we present results of simulation studies to compare performance of our scheme against other existing approaches. In general for applications requiring autonomous adaptation, we show that our two-tier reinforcement learning based scheme on average is about 90% more efficient than traditional resource management schemes like static scheduling while maintaining significant accuracy/performance*

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; C.4 [**Computer Systems Organization**]: Performance of Systems—*Design studies;* D.4.1[**Operating Systems**]*:* Process Management*—Scheduling*

General Terms: Design, Performance, Algorithms, Measurement

Additional Key Words and Phrases: Wireless sensor network, energy and resource management, scheduling, reinforcement learning, tracking.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. INTRODUCTION

Wireless sensor network (WSN) nodes are remarkably constrained in terms of their resources viz. energy, computational power and radio bandwidth. WSNs normally operate in uncertain and dynamic environments where the state of the system changes considerably over time. Moreover, new sensor nodes join or existing nodes move out at any time. WSN applications need to cope with such dynamicity and uncertainty inherent in sensor networks, while simultaneously trying to achieve application’s requirements for QoS and optimization goal. Consequently, adaptive resource management is a key to any successful middleware solution enabling such applications.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Authors’ addresses: Kunal Shah, Department of Computer Science and Engineering, The University of Texas at Arlington, TX 76019. E-mail: [kunal\_u\_shah@yahoo.com](mailto:kunal_u_shah@yahoo.com) ; Mohan Kumar, Department of Computer Science and Engineering, The University of Texas at Arlington, TX 76019. E-mail: [mkumar@uta.edu](mailto:mkumar@uta.edu).

Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Permission may be requested from the Publications Dept., ACM, Inc., 2 Penn Plaza, New York, NY 11201-0701, USA, fax: +1 (212) 869-0481, [permission@acm.org](mailto:permission@acm.org)

©

Resource management includes initial sensor-selection and task allocation as well as runtime adaptation of allocated task/resources. There are many proposed middleware solutions that have advocated strong need for proactive adaptation of resources [4,13,22], but there are only few that have actually tried to resolve the issue of enabling adaptive resource management for WSN applications. This problem of resource management/adaptation can be described as follows:

*Given application structure, QoS requirements and current system state, what is the best way of allocating tasks to resources so that a given system-wide, application-driven, global parameter is optimized.*

In the above, the application structure is in the form of underlying tasks and their interactions. The QoS requirements include such constraints as latency, reliability, coverage etc. while the current state of the system is defined by parameters like mobility, energy availability, and neighboring nodes. The parameters to be optimized include energy, bandwidth, and network lifetime.

We illustrate above resource management problem using a simplified object/entity tracking application. Object tracking application can be considered to consist of following tasks: 1) *sample*- sense the environment (e.g. signal strength of a moving object). 2) *transmit* (Tx)- transmit a message to next hop towards the base-stations. 3) *receive* (Rx)- turn radio to receive mode to listen for incoming messages. 4) *aggregate*- aggregate two or more local and remote same target readings into single reading (e.g. data triangulation for better position estimation or mapping to a known track or simply ‘last value’ aggregation function). 5) *sleep*- put CPU and radio in sleep mode to minimize battery consumption. State representation may consist of the following variables: have one or more neighbors, successful in recent sampling, successful in recent receive, signal strength (or quality of reading). QoS requirements here may include quality of signal, tracking coverage area as well as maximum allowed latency. Our optimization goal in this case is to optimize energy usage among all sensor nodes. Thus 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 coverage and latency requirements of the application.

There have been some research works to study/address the above problem - these can be divided into the following categories:

*Rule/Predicate logic based* *[3,9]:* Here a set of rules are pre-programmed on individual nodes. A rule is fired if all conditions/parameters included in rule predicate evaluates *true* and this may result in task adaptation. This is a simple technique but requires that all state conditions be known in advance, when adaptation might be necessary. Also it can get very complex with large number of nodes and high dynamics, when the system state is changing at a high rate.

*Constraint-satisfaction based [5, 6, 22]:* Here, the problem is defined in terms of constraint-satisfaction and sometimes it is reduced to linear programming with objective function consisting of optimization parameters under given constraints (application requirements). Due to the complexity of the system, it is sometimes not possible to reduce our resource adaptation problem into a linear programming problem without making unreasonable assumptions. Also it doesn’t always allow the distribution making it impractical in large scale WSNs.

*Agent negotiation/Auction based [8]*: This mainly includes multi-agent systems with agents being able to negotiate with each other in order to determine the best allocation. The system consists of one or more mediators responsible for negotiations. Although this approach can lead to efficient resource management, communication and computational resources required for these negotiations may be sometimes prohibitive for their implementation.

*Utility based [1]:* Here the purpose is to define a utility function mapping optimization parameter over number of participating nodes to a real value and maximize these functions under given constraints. This can further take the form of the linear programming problem, but doesn’t allow easy distribution of task management process.

Even though, each of the approaches mentioned above can provide efficient resource management, they suffer from some pitfalls as described. None of these techniques try 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 sensor networks. Therefore, a framework that can enable large set of applications with autonomous adaptation and minimum communication overhead is required.

In this article, we advocate the use of reinforcement learning to address the issue of dynamic resource adaptation in WSN. We have used a bottom-up approach where each sensor node is responsible for task selection instead of top-down approach conventionally used by other middleware solutions. This bottom-up approach using reinforcement learning allows development of autonomous WSN applications with real-time adaptation, minimal or no centralized processing requirement for task allocation and minimal communication overhead. In order to make sure that system is actually meeting the global application goals and is not just acting randomly, we used two-tier learning: micro-learning as used by individual nodes to self-schedule their tasks and macro-learning as used by each set of nodes (e.g. involved/connected in a data-stream) to steer the system towards application goal by setting/updating rewards for micro-learners. Each micro-learner uses Q-learning as their independent RL algorithm, while macro-learners uses Collective Intelligence (COIN) theory to steer system towards application’s global goal.

The rest of the article is organized as follows. Section 2 gives background regarding used techniques viz. Q-learning and COIN theory. Section 3 details our Distributed Independent Reinforcement Learning (DIRL) approach consisting of micro-learners only along with its applicability and shortcomings. In Section 4 we describe how concepts from COIN theory can be utilized for proper guidance of micro-learners to ensure global optimality and overcome shortcomings of DIRL. Section 5 shows how our resource management framework can be applied to some real-world applications like object/entity tracking, intrusion detection and health monitoring. In Section 6 we present performance analysis of our two-tier RL based framework. Section 7 concludes the paper with directions to future work.

2. BACKGROUND

In this section, we will briefly describe Reinforcement Learning (Q-learning) and COIN theory concepts 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 [16]. Agent has to try out different actions in order to learn what actions yield the most reward. RL is very useful for interactive/online learning in dynamic uncertain environments. Wireless sensor networks can be modeled as a multi-agent system (MAS) with each sensor represented as a goal-oriented agent. Standard RL techniques (e.g. Q-learning) can be applied directly to MAS. In MAS, reinforcement learning can take any of the following two forms [2]: a) Independent learners- where the learning algorithm is applied in a classic sense ignoring the presence of other agents; and b) Joint Action Learners (JAL) - where agents try to learn in conjunction with other agents by coordination through game-theoretic approaches and considering their joint actions. Use of JAL involves much more communication and processing overhead and may be an over-kill for WSN. Also, Claus and Boutilier [2] have shown that “even though JAL have much more information at their disposal, they do not perform much differently from independent learners in straightforward application of Q-learning to MAS”.

Q-learning [17] is a form of model-free reinforcement learning. Q-learning is quite simple, demands minimal computational resources and doesn’t require a model of the environment in order to operate. Hence it is ideal for implementation on resource-constrained sensor nodes. It 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 sum of immediate reward ‘*r*’ and discounted utility of resulting state ‘*s*’’ after executing task ‘*t*’, i.e.

*Q(s,t) = E [r + γ e(s’) | s, t]* - (i)

where *e(s’) = Maximum Q(s’,t)* over all tasks ‘*t*’. Note that the expected value above is conditional upon being in state *‘s’* and performing task ‘*t’*.

As ‘Q-learning’ is done online, the above equation cannot be applied directly as stored utility values may not have converged yet to final values. Hence, in practice, Q-learning is used with incremental step updates as given by the following:

*Q(s,t) = (1- α) Q(s,t) + α ( r + γ e(s’))* - (ii)

Here ‘*α*’ is a learning-rate parameter in between ‘0’ and ‘1’. 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. This means setting ‘*α*’ equal to 1 will make agent ignore all previously learned utilities resulting in single-shot learning. ‘*γ*’ is a discount-factor and also varies from ‘0’ to ‘1’. The higher the value, the greater the agent relies on future reward than the immediate reward. In our experiments we have used *α*=0.5 and *γ*=0.5.

An important aspect of 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 to 0.5) or can be derived using some other heuristics like starting with a high value and gradually decreasing for example using the Boltzmann equation [16].

A COIN [REF] is a large multi-agent system where there is a well-defined ‘world utility’ function which rates the behavior of the entire system and where there is little to no centralized control. Each agent in the MAS is ‘selfish’ and runs a RL algorithm. System’s global behavior hence is the collective effect of individual agents each modifying their behavior using RL algorithm. COIN theory addresses the following design problem:

*Given the individual agents are maximizing their own (local) utility functions (e.g. Q-learning), how to design these local utility functions to ensure optimum world utility, given that individual agents attempt to maximize their own utilities and that agents do not frustrate other agents resulting in lower world-utility.*

The RL algorithms at each agent that aim to optimize their local utilities are called *microlearners*. The learning algorithms that update the agent’s utility functions are called *macrolearners*. COIN theory uses game-theory concepts to devise a methodology for designing/updating local utility functions at each agent so that system will approach near-optimal values of the world utility. We will address concepts used by COIN as well as its application to WSN in Section 4.

3. distributed independent reinforcement learning (DIRL)

The main idea of DIRL is to allow each individual sensor node to self-schedule its tasks and allocate its resources by learning their usefulness (utility) in any given state while honoring application defined constraints and maximizing total amount of reward over time. The advantage of using independent learning is that no communication is required for co-ordination between sensor nodes and each node selfishly tries to maximize its own rewards.

We make the following assumptions in designing our model: i) Each node is able to perform only one task at a time; and ii) A node is only allocated with a set of tasks it’s capable of executing, i.e. task deployment depending on sensor node type (in case of heterogeneous network) is already done.

Global optimization parameter that application is interested in can be specified in terms of individual task rewards at a sensor node. Thus, if an application needs to optimize energy usage, each task’s reward function can be a combination of task outputs as well as energy consumed for performing that task.

3.1 Applying RL to WSN Resource Management

In order to apply RL to our resource management problem, we will need to define our problem in terms of the elements of RL. Elements of RL system and their mapping to our problem are as follows:

*Agent:* As mentioned earlier each sensor node corresponds to an agent in multi-agent reinforcement learning (MARL).

*Environment*: The world surrounding the sensor node, with which it interacts with.

*Action:* An agent’s action in our case is the application task to schedule. 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 cycle. For example an agent may have the following set of actions: transmit, receive, sample, alarm, actuate, aggregate etc.

*State:* A set of application defined variables and system variables constitute a node’s state. For example, one can include system variables like number of neighboring nodes, remaining energy, mobility, capability for outbound/inbound communications etc. Application specific variables such as sensor readings signal strength etc. may also be part of the *state*. Number of states in a system can grow exponentially with increase in state variables and most of the time it is not practical to enumerate all possible states in advance. DIRL uses weighted hamming distance as a method to classify group of similar states and thus reducing number of system states that it needs to keep track of.

*Policy:* An agent’s policy determines what action it will take in a particular state. In our case, this policy determines which task to execute in the perceived sensor state. We have defined policy that consists of predicates as well as exploitation/exploration strategy, as will be discussed later in this section.

*Reward function*: It provides a mapping of agent state and corresponding action to a reward (typically, a real number) that contributes to the utility. This is how an agent is guided through its learning process with an objective to contribute to the goal. Each agent’s goal is then to maximize total reward over time. In our scheme, this reward function needs to map application defined optimization parameter (e.g. energy usage, bandwidth utilization, network lifetime etc.) into a numerical reward, as this is the goal that each agent tries to achieve. Each task implements a simple reward function that gives the amount of reward (positive or negative) obtained during each execution of that task. For example, if the task is *receive*, then its reward will be a function of number of messages actually received during its execution as well as amount of resources consumed that it is trying to optimize. This reward function can be as simple as returning a pair of constant values or may be complex considering various optimization constraints.

*Value function*: This is an important aspect of any RL system. It defines what is good for an agent over long run including current as well as possible future states and not just immediately as described by reward function. But essentially, it is built upon reward function values over time and hence its quality totally depends on reward function. We use Q-learning, a form of RL that has intrinsic value function defined.

Figure 1 illustrates the above elements along with interactions among them.



Figure 1. Elements of RL system applied to DIRL

For exploration/exploitation trade-off in DIRL, we have used a simple heuristic where exploration probability at any point of time is given by

*ε = minimum (εmax , εmin + k\* (Smax-S)/Smax)*  -(iii)

where *εmax*and*εmin* define upper and lower boundary for exploration factor respectively, while *Smax* represents maximum number of states (as obtained from application) that DIRL will try to map and *S* represents current number of states already known. Thus the above heuristic allows initial exploration with a higher rate and gradually decreasing over time as DIRL is able to map/discover more states. ‘*k*’ is a constant that can be tuned to control the effect of unexplored states. We have used *k*=0.25, *εmin*=0.1 and *εmax*=0.3 in our experiments. Note that some minimum exploration is required at all times to allow a node to dynamically reconfigure in case of environmental changes.

In DIRL, each node chooses a task to execute at each time cycle either by exploitation or exploration. But, all tasks may not be executable at all times. For example, *aggregate* task may not be able to execute if there are no readings available to aggregate. Also DIRL needs to honor certain application constraints like latency, quality of readings etc., while scheduling tasks. In order to achieve this, DIRL allows each task to be associated with an applicability predicate that needs to be evaluated *true* in order for that task to be executed. Thus a task is chosen for execution only if its applicability predicate returns *true*. Again with *aggregate* task, if application has constraints on maximum latency of a reading, then one can include this constraint in applicability predicate for *aggregate* task. Figure 2 shows the flow diagram of task scheduling using our exploration/exploitation policy along with task’s applicability predicate at a particular time-step **‘*τ*’**.

Another important characteristic of a RL system is how it handles temporal and structural credit assignment problem [11,15]. Temporal credit assignment is the problem of propagating reward backwards in time while structural reward is related to propagating reward spatially across states in order to define notion of similar states. Q-learning and its extensions does provide support for temporal credit assignment problem dealing with delayed reward [17], but for simplicity, in this paper we will restrict to only simple single step immediate reward. It is important to resolve structural credit assignment problem though, as otherwise each node will end up with enormous state-space to work with, which is not practical for wireless sensor networks. DIRL uses simple weighted hamming distance between two states in order to resolve structural credit assignment problem. While defining a state representation in the form of system and application variables, application also specifies how much weight each variable carries. This weight specifies to DIRL, how much change in that variable is important for defining state difference. Thus if an application’s state representation consists of variables *V1, V2,,…,Vn* with corresponding weights *W1, W2,,…..,Wn*, then DIRL uses this information to determine if two given states ‘*s1*’ and ‘*s2*’ are similar or not, by calculating hamming distance between them as follows:

*H(s1-s2)= W1\*|(V1(s1)-V1(s2))| +W2\*|(V2(s1)-V2(s2))| + ….*

*+ Wn\*|(Vn(s1)-Vn(s2))|*  -(iv)

If this hamming distance is less than a threshold value then two states *s1* and *s2* are considered to be similar and has only one entry in Q data-structure.



Figure 2. Flow diagram for task selection process at time-step ‘τ’

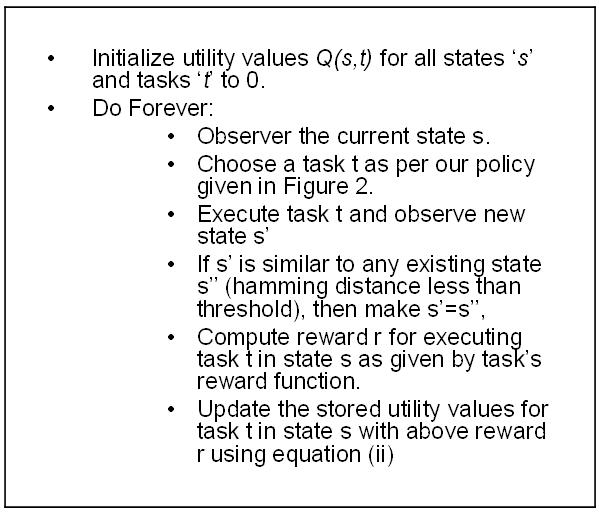


Figure 3. Algorithm performed by DIRL

DIRL needs the following as inputs from the application:

1. Variables of interest to the application and QoS required for each variable[[1]](#footnote-2).
2. Task graph describing application specific tasks that needs to be performed on sensor network infrastructure. Task graph mainly contains set of task nodes connected in a specific order and is passed on to all participating sensor nodes in the system. Most common tasks may involve *sample*, *route* and *sleep*. Task graph may also include tasks involving in-stream (while data has not reached destination) and distributed data conversions and aggregations e.g. *average*, *triangulate* etc.
3. Associated with each task, an applicability predicate incorporating any application specific QoS or other constraints and reward functions guiding towards optimization goal. The reward functions are designed so that obtained reward is a function of expected price/reward given as an input to the function. The separation of expected price from reward function allows dynamic external tuning of reward function and hence the overall system behavior by changing the expected price part (only) of the reward function over-the-air [12].
4. A state representation consisting of a combination of system and application variables, along with their weight that will be used in determining weighted hamming distance to aggregate similar states.
5. Maximum number of states that DIRL should try to explore. This gives an upper bound on number of states in the system and 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 existing set of similar states.

Once the above information is available, DIRL performs the algorithm given in Figure 3.

3.2 Consideration for Global Optimality

The main advantage of using independent learning in DIRL is that no communication is required for co-ordination between sensor nodes and each node selfishly tries to maximize its own rewards. This works fine when each node in WSN application is acting on its own and doesn’t need to co-operate or compete with other nodes. In other words, if all nodes are acting independently and their actions do not affect others, then any increase in a node’s utility cannot decrease anyone else’s utility and hence will always increase world (system-wide) utility which is merely sum of all nodes utilities over all times. As we will see in the next section, such a system is sub-world factored and will eventually attain a Pareto-optimal point and hence leads towards our system-wide optimization goal.

But most of the real-world WSN applications need some sort of co-operation among sensor nodes and hence nodes cannot work independently. In this case, increase in utility of individual node may result in reduction of other node’s utility and hence may not increase world utility. It is also possible that such system can lead to the Tragedy of the Commons (TOC) phenomenon or Braes’ Paradox [TODO], wherein individual’s selfishness leads to significantly lower global utility. Such phenomena can be avoided by carefully designing agent’s utility functions as well as constraints under which agent performs task selection. In other words, we need to make sure that individual’s utility is ‘aligned’ with the global (or world) utility, i.e. any increase in agent’s private utility because of its action will also result in increase of world utility. A real-world example of this would be human economy, wherein each individual tries to maximize his private utility in the form of income, career advancement etc. If there are no constraints over how individuals can operate and maximize their utility, he can operate at cross purposes to other human beings and thereby leading to the downfall of the global utility (say GDP). Here government regulations act as necessary constraints and modifications to human utility function in order to ensure its alignment to global utility. Regulations are designed such that any increase in human utility will also cause increase in global growth of the economy.

In our framework, we don’t have to deal with virtues of trust and honesty among sensor nodes as the design of utility functions of all nodes in the system is in our hand and hence they can be ‘coded’ to be honest. Nevertheless, the problem of attaining sub-optimal system wide utility is still required to be addressed, even if nodes are honest. This is mainly because of imperfect/partial knowledge that each node has for the rest of the system. This partial knowledge of one node can cause it to perform an action which may not benefit the overall system. Hence, key to achieving higher global utility lies in the design of private (individual node) as well as global utility functions such that they are aligned. In this paper we investigate the adaptation of COIN theory to address this critical problem in sensor systems.

Another drawback of using DIRL in real-world application is determining reward functions/price settings. It is difficult to investigate different aspects of system dynamics and choose reward settings for each resource/task. Also these settings need to be changed on the fly whenever overall system state and/or application requirements change. Hence, hand-tuning of reward-settings is not acceptable. 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 micro-learners utility function as and when application state/requirement changes and always steer them toward global optimization goal.

In the next section, we present further details about COIN theory and how we have adapted it to the problem of resource management in sensor systems in order to steer system towards higher global utility and to provide some type of guarantee towards achievement of system goal.

4. Application of collective intelligence based macro learning

The problem of resource management in WSN is setup in terms of COIN in order to apply its concepts. Consider a WSN system, consisting of N nodes, evolving across a set of discrete, consecutive time steps τ ∈ {1,2,3,.}. Let Ϛnτ be an element of a vector space Znτ,, representing *state* of a node ‘n’ 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, Ϛτ ∈ Zτ denotes a global state of our system, combining actions/variables from all nodes, at a particular time step τ, while Ϛ ∈ Z is the vector of global state at all times. Ϛ can also be referred as world-line [13] in space Z over all time steps. The goal of our framework is then to determine an optimal world-line Ϛ by maximizing some system-wide global utility function of Ϛ i.e., G(Ϛ) and then steering system along that world-line. Each node ‘n’ in our framework is trying to maximize its private utility function say qn(Ϛ). Thus, we need to show here how and under what conditions, framework can determine optimal Ϛ, given each node trying to maximize its private utility function qn(Ϛ) (Q-learning value function).

4.1 Concepts of COIN theory in WSN Resource Management Context

The salient features of COIN theory [13, 15] along with their applicability to the resource management problem in WSN are described below:

* A *collective* is defined as a multi-agent system wherein each agent is adaptively trying to maximize its own private utility function, while at the same time there is system-wide performance criteria defined to rate the behavior of the entire system. Thus, our system of wireless sensor networks comprising individual sensor nodes adaptively trying to maximize their utility function, in order to achieve system-wide goal is a *collective*.
* Most of the collective system focuses on *forward* problem of how the localized attributes induce a global behaviour and thereby determining system performance. COIN on other hand addresses the *inverse* problem of designing a system to induce behaviour that maximizes world utility. This is done by designing either private utility functions or incentives to private utility functions. Likewise, our objective is to design reward/utility functions used by individual sensor nodes in the WSN so that system-wide utility can be maximized.
* *Subworlds* are sets making up an exhaustive partition of agents. For each subworld, *w*, all agents in that subworld share the same subworld utility function g*w*(Ϛ) as their local utility functions. Accordingly, consider each subworld to be a set of agents that collectively have the most effect on each other. In this situation, by and large, agents cannot work at cross-purposes, since all agents that affect each other substantially share the same local utility. WSN applications mainly being data-centric, a chosen subworld is a set of sensor nodes involved in a *data stream* i.e. from data-source to data-sink. For example, all nodes involved in a particular data stream – from data sensors, to aggregators and collectors will be part of single sub-world as they all have immediate and considerably high effect on each-other and hence share single utility function. This sub-world definition suggests that sub-world formation is dynamic and can change along with the state of the system. Fig. 1 marks three such data-stream subworlds in a sensor network comprising of 10 nodes in an object tracking application.



Fig. 1: Data-stream subworlds in WSN for object tracking application

* Associated with sub-worlds is the concept of a (perfectly) *constraint-aligned* system. That is a system in which any change to the state of the agents in sub-world *wi* at time *t* will have no effect on the states of agents of sub-world *wj* (*i* ≠*j*)at times greater than *t*. Intuitively, a system is constraint-aligned if no two agents in separate sub-worlds affect each other, so that the rationale behind the use of sub-worlds holds. Most of the real-world systems are not perfectly constraint-aligned and same is the case with WSN. A change of state in a node involved in one data-stream (and hence one sub-world) may affect state of other node in near-by data stream. But the effect here will be probably much less compared to effect on node in the same data-stream. Nevertheless, this is an assumption that needs to be experimentally validated.
* A *subworld-factored* system is one where for each sub-world *w* considered by itself, a change at time τ to the states of the agents in that sub-world, when propagated across time, results in an increased value for g*w*(Ϛ) if and only if it results in an increase for G(Ϛ). Mathematically, a system is subworld-factored if following holds true for all pair of states Ϛ and Ϛ’ that differ only for subworld *w*:

g*w*(Ϛ) ≥ g*w*(Ϛ’) ⬄ G(Ϛ) ≥ G(Ϛ’)

For a subworld-factored system, the side effects on the rest of the system of *w*’s increasing its own utility do not end up decreasing world utility. In our problem, if we model a system where each sensor node, as a subworld, selfishly tries to maximize its private utility, then system will not be subworld-factored (assuming sensor nodes are not totally independent). This is because 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 using appropriate subworld utility function (e.g. Wonderful Life Utility Function [13]) g*w*(Ϛ) will not reduce world utility because of relative independence with other data-streams for the period of sub-world’s existence. Hence such a system can be considered as subworld factored.

Another requirement of application of COIN theory is to have private utility functions with higher *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 team game (where reward of each node is same as that for all nodes in the system at any time step), is much less than those of self-only utility functions. Thus, 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 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 number of nodes in single data-stream is just a fraction of total number of nodes in WSN.

Wonderful Life [13] utility function plays an important role in designing a subworld-factored system because of the fact that a constraint-aligned system with wonderful life (WL) subworld-utilities is subworld-factored. If CL*w*(Ϛ) is defined as vector Ϛ 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*(Ϛ)= G(Ϛ) - CL*w*(Ϛ) - (v)

This definition of WL utility is same as setting WL utility to world utility when considering that subworld *w* had never existed. Thus a subworld’s utility 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 DIRL’s private utility functions i.e. qn( Ϛ).

*COIN* theory [13, 15] 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. This Nash Equilibrium point is also the Pareto optimal point of the system. From the above description, we can also 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 upsets like 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.

4.2 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 top-down approach (where some central entity dictates nodes what task to execute) used by many other middleware solutions [4, 16]. 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 resource management framework is to determine best allocation of task to sensors/resources so that application defined optimization goals, such as energy savings, network lifetime longevity, bandwidth preservation etc., can be achieved while simultaneously honouring application’s QoS metrics. QoS may be defined in terms of quality of measured variables (sensed and processed data) or other application constraints like latency, reliability etc. Hence, system’s global utility function G(Ϛ) should be a function of success towards achieving optimization goal as well as application’s QoS metrics which in-turn are based on variables of interest to the application. Also G(Ϛ), which represents global utility over all time, can be expressed here as a sum of rewards ∑τRτ(Ϛτ­), where Rτ is global reward and Ϛτ is global state at time-step *τ*. Thus, Rτ are temporal translations of one another. Thus,

G(Ϛ) = ∑τRτ(Ϛτ­) - (vi)

As global utility function may take many forms and is application specific, we allow application to define Rτ(Ϛτ­) given the current state of the system as represented by measured variables (data) and optimization parameters. All optimization parameters are represented in the form of running-sum of numerical rewards attached to each data packet. It is also possible to provide generic implementation of Rτ(Ϛτ­) based on QoS requirements specification and total of rewards from all data-streams.

Each micro-learner uses Q-learning as in DIRL and hence their private utility function qn(Ϛ) is a Q-learning value function as given by (ii).

Macro-learners on the other hand use 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 unit). From (i) and (ii), wonderful life reward of each agent (node) part of subworld *w* at time-step *τ* is given by:

g*wτ*(Ϛτ)= Rτ(Ϛτ) - CL*w*(Ϛτ) - (vii)

In this case, CL*w*(Ϛτ) is the world reward Rτ(Ϛτ) after removing all data values that have been reported by data-stream (subworld) *w.* This nulls out the effect of data-stream *w* on the world reward, but considers the actual contribution towards improving world reward Rτ(Ϛτ). Suppose data-stream *w* is providing values of a data variable which by itself is not significant, but has higher effect on overall global application goal, *w* gets 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, hence 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 manage resources using this COIN based framework, we made following extensions to the application input for DIRL described in section 3.1:

1. Instead of hand-tuning expected prices associated with reward function of each task (which we found very difficult to do given various system dynamics), in this scheme, expected price is set by macro-learner using WL utility of its sub-world. Again we have used numerical price here to allow macro-learner to update node’s private utility functions without incorporating new code.
2. Application also provides a global reward function Rτ(Ϛτ­) which returns global reward for a time-step τ, given the current state of the system as represented by measured variables (data) and optimization parameters.

Each sensor node in the system has two agents: 1) micro-learner which is self-contained, trying to maximize its private utility using local information only and 2) macro-learner which is part of a subworld containing other sensor nodes linked in a data-stream and sharing same utilities. Once application input is available, system enters into *Initialization* phase. Note here that initially there are no learned utilities available with either micro-learners or with macro-learners and hence system needs to go through some sort of initialization. Initialization is possible using any of the following three options:

1. Self-exploration and system learning: Reinforcement learning based system always uses some balance of exploration (trying out some random actions in search of better rewarding actions) and exploitation (choosing actions based on build utilities) to build up its knowledge base. During initialization phase, the rate of exploration will need to be higher compared to that of exploitation. Hence decisions made by the system during this phase will be more random and may not lead towards a system goal. A given WSN system and the application using it may be flexible enough to allow such initial self-learning phase and thereby building utilities over time. In this type of system, self-learning will be the viable choice.
2. Using domain knowledge: It is possible to incorporate domain expertise 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 initial estimate of task’s worth for given sensor node. These values can also be determined using application model simulation using self-exploration as described in first option above and then fed to the deployed system. As mentioned earlier, micro-learners have very high learnability and hence can build their utilities quickly.
3. Employing an available sensor-selection scheme: We can also utilize any of the various sensor-selection techniques as published in related work e.g., MidFusion [1], MiLAN [4] etc. Results from these techniques can be used to initialize macro-learners with expected price for individual sensor nodes.

After initialization, micro-learners and macro-learners have knowledge base to start decision making and the system is then considered to be in *Normal* operating phase. During normal phase, micro-learner tries to maximize its private utility function by using Q-learning based algorithm as given in Fig. 2. Micro-learner uses either exploration or exploitation for task selection at each time step τ based on exploration factor and uses hamming distance between two states as a criterion to distinguish separate states allowing reducing state space of the sensor node [10]. Micro-learner gets an immediate reward after task execution at each time-step which it uses to update its Q-learning value function as well as to update a running sum of reward on the data-packet that it acted on. As the reward obtained is a function of application’s optimization goal (e.g. minimizing energy usage), this running sum of reward on data-packets will be a measure of how well each data-stream is performing towards application’s goal. A tuple consisting of chosen task ID and data-packet ID at time step τ is recorded with macro-learner so that macro-learner can provide future feedback on this task execution.



Fig. 2: Algorithm used by each micro-learner

Data-sink (base-station/controller) is responsible for determining wonderful-life reward g*wτ*(Ϛτ) (given by (iv)) for time-step τ for each data-stream. Here global reward Rτ(Ϛτ­) is calculated using application specific or generic global reward function given data streams output (measured variables) and total of running-sum of rewards from all data-streams (acting as a measure of optimization parameters). CL*w*(Ϛτ) is also calculated using same global reward function but using all data-streams other then *w* and discarding reward obtained from *w*. Value of g*wτ*(Ϛτ) determined as above for a data-stream *w* is then pass down to participating nodes in *w* along the reverse path of the data-stream. As each macro-learner gets its reward from data-sink, it may arrive after few time-steps. Hence macro-learner maintains a recent history of tuples recorded by micro-learner (a tuple for each time-step), until it receives reward for that time-step. Received reward is matched with recent history using packet id and is used to update utility value (expected price of micro-learner) for the associated task. This process is demonstrated in Fig. 3.



Fig. 3: Sequence Diagram for Macro-learning process



Fig. 4: Overview of activities between application and WSN using two-tier reinforcement learning

Figure 4 gives a high-level overview of our framework along-with interactions with application. As part of application deployment, task graph and associated application constraints as well as reward and cost functions are dispersed on to the nodes of the sensor network. Application also provides global reward function to applicable data sinks. Figure also shows optional initialization of sensor node’s local utilities. Application can provide variables of interest with their required QoS at any state change. From this point onwards, each node takes on the responsibility of self-scheduling their tasks and allocating resources based on learned local utilities. All data-streams are evaluated for WL reward at the end of each time-step. WL reward is next distributed to all the nodes in respective data-stream (if considerably different from previous reward). Sensor nodes participating in data-stream updates their local-utility functions based on the global WL reward.

The simple set of micro-learner and macro-learner provides each sensor node the capability to self-schedule tasks, while making sure that the overall system is being steered towards its optimization goal. This scheme allows a sensor node to self-adapt to system dynamics and uncertainty inherent in the WSN. Micro-learner adapts to local state changes (e.g. low battery, neighbor change, nearby target etc) immediately while on the other hand macro-learner provides adaptation at the global level with change in global state or application requirement (e.g. change in QoS of data variables, addition/removal of sensor nodes, etc.). Any such global change ensues a change in wonderful-life utility of macro-learner which changes utilities learned by micro-learner in that direction. The changing reward function of micro-learners may invalidate learned utilities (Q-values) and hence Q-values may not converge faster to equilibrium [12]. But it is possible to discard learned utilities and start over again whenever reward function is updated. Also we are more interested in adaptation as well as convergence of the system as a whole rather than convergence of individual micro-learners.

5. Implementing Real-World Applications

Our design of resource management framework has been motivated by the need for generalization so that various classes of WSN applications can 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 the real-world WSN applications can be easily implemented over our two-tier RL based resource management framework.

* 1. Object Tracking

Object tracking application can be deployed on top of heterogeneous WSN to track one or more objects of interest. This may be to provide surveillance for environmental monitoring or for a battlefield. Depending on the usage, application may have minimum sensor coverage area and lifetime requirements. Application may not need redundant information provided by overlapping sensors. Also tasks performed by each sensor node (e.g. sampling, routing etc) can be tuned based on current state of the system, e.g. presence of objects. Hence, by means of efficient and continuous adaptive resource management over time, it’s possible to allow sensors to preserve energy while still meeting application’s requirements.

Tasks involved in an object tracking application have already been described earlier in section 1. Table 1 shows these tasks with associated reward functions. Here we want to optimize energy usage among all sensor nodes as can be seen from the reward function which penalizes each task with the amount of energy consumed. Note that expected price and reward functions are designed in order to reward node only if task execution results in success. Thus if a node schedules task *receive*, then node will get positive reward only if one or more messages are received in that time step, otherwise it will receive penalty proportional to energy consumed during the time step. Our state representation consisted 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 threshold hamming distance of 1.0 and maximum number of states was set to 5.

Global reward function for object tracking application is also shown in table.

Table 1. Implementation of WSN Applications

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Object Tracking** | **Intrusion Detection** | **Health Monitoring** |
| Variables | signal strength (SS), area coverage, target(s) position/track | Security threat confidence level, area coverage | Heart rate, respiratory rate, blood pressure |
| QoS Requirements | SS>threshold &  coverage &  Delay<threshold | Confidence level > threshold & k-coverage | Quality of measured variables > threshold |
| Types of Sensors | Acoustic, seismic, video | Motion detectors, Biometric-reader, RFID, video | ECG, blood pressure, blood flow, EMG etc. |
| Tasks and Reward Functions | |  |  | | --- | --- | | *Aggregate* | (no of samples aggregated\* price) – energy spent | | *Tx* | (no of Tx \* price) –energy spent | | *Rx* | (no of Rx \* price) –energy spent | | *Sample* | (no of sensed events with SS>threshold\* price) –energy spent | | *Sleep* | price–energy spent | | |  |  | | --- | --- | | *Tx* | (no of Tx \* price)  –energy spent | | *Rx* | (no of Rx \* price)  –energy spent | | *Sense* | f(Confidence level)\*price–  energy spent | | *Sleep* | price–energy  spent | | |  |  | | --- | --- | | *Tx* | (no of Tx \* price)  –energy spent | | *Rx* | (no of Rx \* price)  –energy spent | | *Sense* | f(Quality of  measured variable)  \*price  –energy spent | | *Sleep* | price–  energy spent | |
| Global Reward Function Rτ(Ϛτ) | For each target at time-step τ :  f(SS, coverage, tracking delay) – total cost of data acquisition | f(threat confidence level, coverage) – total cost of data acquisition | f(quality of measured variables) – total cost of data acquisition |

We would like to point out here that it is also possible to apply RL technique hierarchically in order to schedule sub-tasks for any given task. For an example for the sample task, consider a sensor node with radar having three heads [8] and at any time it can activate only one of them to detect target. These three heads can be considered as three sub-tasks for ‘sample’ task. Thus micro-learner can be used to learn utilities of activating each head in different state and thus determining which one will be the best in a given state. Similarly ‘transmit’ task may need to choose one neighbor out of ‘n’ next hop neighbors towards base-station. By applying micro-learning to sub-tasks, a node can learn which neighbor provides highest utility for successful transmission of message to base-station.

* 1. Intrusion Detection

In intrusion detection application, variety of sensors e.g. motion detectors, RFID, video etc., may be utilized for detecting an intruder; each providing different level of confidence and can have different cost of data acquisition. Large number and types of these sensors may be deployed in a given WSN for redundancy as well as flexibility of choosing the required sensory information. Sensory information required by the application depends on the state, for an example, on detecting high probability of intrusion; application may want to utilize video cameras for highest quality of information. Hence again by properly managing sensor resources, it is possible to optimize their usage while meeting the requirements of application. High level tasks and reward functions for intrusion detection application are similar to that of object tracking application and are detailed in table 1.

* 1. Health Monitoring

For health monitoring application also a large variety of sensors can be used each providing one or more health-related variables with different quality as well as cost. For an example, heart rate can be measure by ECG, blood pressure monitor or blood flow monitor [MiLAN]. But accuracy and quality of each of them is different and so does the cost of obtaining heart rate. Intelligent resource management can help in choosing less costly sensors in say normal health state while triggering expensive but highly accurate sensor in abnormal state. Input from such health monitoring application as required by our resource management framework is described in table 1.

6. Simulations and Analysis

We have chosen object/entity tracking application for the analysis of our framework, as it is widely used in WSN literature concerned with middleware developments [8, 12, 18] and can help with comparative analysis.

Next, we present comparative performance analysis of DIRL against three other schemes: a) STATIC-where each node performs a fixed set of tasks repeatedly without trying to do any dynamic adjustments, b) RANDOM-where each node performs a task randomly chosen from uniform distribution each time, and c) SORA [12]-which uses simple heuristic based reinforcement learning. We will present our results for a 10 nodes scenario with sensor nodes up to two hops away from base-station and random target movement over a grid of size 500 ×500m. Figure 6 compares algorithms in terms of number of executions of each task. Nodes in DIRL execute *sleep* task more than 45% of time even in the scenario where target node is continuously active and in range throughout the simulation.

As optimization goal of our framework is defined in terms of rewards, best way to measure its performance is using metric of average reward over time. Figure 7 shows average reward over time for all the four algorithms studied. We can see here that DIRL outperforms all other algorithms in terms of maximizing reward followed by STATIC and SORA. RANDOM as expected earns the lowest reward.

**Table 2. Performance comparison of DIRL**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **DIRL** | **STATIC** | **RANDOM** | **SORA** |
| **Energy Efficiency** | 51.94 | 29.42 | 20.84 | 44.22 |
| **Success Rate in *sample*** | 51.18 | 42.48 | 29.06 | 66.14 |
| **Success Rate in *receive*** | 31.85 | 31.75 | 24.91 | 7.23 |
| **Mean Tracking Error** | 5.79 m | 6.36m | 14.03m | 14.49m |



**Figure 6. Total no. of executions for each task type**

Table 2 compares efficiency of DIRL against other algorithms in terms of four different metrics. Energy efficiency is determined as ratio of energy successfully utilized towards packets that arrive at base-station to total energy consumed in entire system. Success rate in *sample* and *receive* is percentage of time *sample* and *receive* tasks received positive feedback. Finally, mean tracking error is calculated over all target readings obtained at base-station for one simulation run. As expected, DIRL performs quite well in terms of energy efficiency and success rate (as it tries to tune itself to favorable tasks). Also in terms of tracking accuracy, DIRL’s performance exceeds that of STATIC schedule which uses about twice as much energy as DIRL. Tracking efficiency of DIRL can also be seen in scatter-plot of data-points received at base-station as given in Figure 8.



**Figure 7. Average reward over time**



**Figure 8. Scatter-plot of tracking error at base-station**

References

[1] J. Byers and G. Nasser, Utility-Based Decision-Making in Wireless Sensor Networks, *IEEE Mobihoc* (2000).

[2] C. Claus and C. Boutilier. The Dynamics of Reinforcement Learning in Cooperative Multi-agent Systems. In Proceedings of the National Conference on Artificial Intelligence, 1998, NUMB 15, pages 746-752.

[3] C. Frank, K. Romer, Algorithms for generic role assignments in wireless sensor networks. In Proceedings of the 3rd international conference on Embedded networked sensor systems, 2005.

[4] W. Heinzelman, A. Murphy, H. Carvalho and M. Perillo. Middleware to Support Sensor Network Applications. IEEE Network Magazine Special Issue, January 2004.

[5] S..Kogekar, S. Neema, B. Eames, X. Koutsoukos, A. Ledeczi, M. Maroti. Constraint-Guided Dynamic Reconfiguration in Sensor Networks. Proceedings of the third international symposium on Information processing in sensor networks, 2004.

[6] B. Krishnamachari, S. Wicker, R. Béjar, C. Fernàndez.On the Complexity of Distributed Self-Configuration in Wireless Networks. Telecommunication Systems, Volume 22, Numbers 1-4/January,2003.

[7] P. Levis and D. Culler. Mate: A Tiny Virtual Machine for Sensor Networks. In Proc of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 2002.

[8] V. Lesser, C. Ortiz, and M. Tambe. Distributed Sensor Networks: A multiagent perspective*.* Kluwer Publishers, 2003.

[9] Tin Liu, Margaret Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. In ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2003.

[10] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks. In *OSDI 2002*, Boston, USA, December 2002.

[11] S. Mahadevan and J. Conell. Automatic programming of behavior-based robots using reinforcement learning. Artificial Intelligence, Volume 55, Issue 2-3, 1992.

[12] G. Mainland, D. Parkes and M. Welsh. Decentralized, adaptive resource allocation for sensor networks. In Proceedings of 2nd Symposium on Networked Systems Design & Implementation, 2005.

[13] P. J. Marron, A. Lachenmann,  D. Minder, J. Hahner, R. Sauter, K.  Rothermel. TinyCubus: a flexible and adaptive framework sensor networks. In Proceedings of the Second European Workshop on Wireless Sensor Networks, 2005.

[14] Project Sun SPOT (Small Programmable Object Technology). <http://www.sunspotworld.com>

[15] R. Sutton. Temporal credit assignment in reinforcement learning.  Ph.D. dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA 01003. Published as COINS Technical Report 84-2.

[16] R. Sutton and A. Barto. Reinforcement Learning: An Introduction. MIT Press. ISBN 9-262-19398-1. Available online at <http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html>.

[17] C. J. C. H. Watkins. Learning with Delayed Rewards. Ph.D. Dissertation, Cambridge University, 1989.

[18] C. J. C. H. Watkins, and P. Dayan. 1992. Q-learning. Machine Learning 8(3):279–292.

[19] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In Proc. the first Symposium on Network System Design and Implementation, March 2004.

[20] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: A neighborhood abstraction for sensor networks. In Proc of the International Conference on Mobile Systems, Applications, and Services (MOBISYS‘04), June 2004.

[21] J-Sim, Java based network simulator. http://www.j-sim.org/.

[22] Y. Yu, B. Krishnamachari, and V.K. Prasanna. Issues in Designing Middleware for Wireless Sensor Networks. IEEE Network Magazine, special issue on Middleware Technologies for Future Communication Networks., January 2004.

1. This is similar to input required for MiLAN [4], but we don’t require sensor QoS graph like in MiLAN as data generated by sensors can provide its quality at runtime. [↑](#footnote-ref-2)