A Framework for Resource-Aware Data Accumulation in Sparse Wireless Sensor Networks

Kunal Shah[a], Mario Di Francesco[a], Giuseppe Anastasi[b], Mohan Kumar[a]

[a] *University of Texas at Arlington*
[b] *University of Pisa, Italy*

Abstract

Wireless sensor networks (WSNs) have become an enabling technology for a wide range of applications. In contrast with traditional scenarios where static sensor nodes are densely deployed, a sparse WSN architecture can also be used in many cases. In a sparse WSN, special mobile data collectors (MDCs) are used to gather data from ordinary sensor nodes. In general, sensor nodes do not know when they will be in contact with the MDC, hence they need to discover its presence in their communication range. To this end, discovery mechanisms based on periodic listening and a duty-cycle have shown to be effective in reducing the energy consumption of sensor nodes. However, if not properly tuned, such mechanisms can hinder the data collection process. In this paper, we introduce a Resource-Aware Data Accumulation (RADA), a novel and lightweight framework which allows nodes to learn the MDC arrival pattern, and tune the discovery duty-cycle accordingly. Furthermore, RADA is able to adapt to changes in the operating conditions, and is capable of effectively supporting a number of different MDC mobility patterns. Simulation results show that our solution obtains a higher discovery efficiency and a lower energy consumption than comparable schemes.

Keywords: Sparse wireless sensor networks, mobile data collectors, reinforcement learning, resource allocation, adaptive algorithms

Email addresses: `kunalbhai.shah@mavs.uta.edu` (Kunal Shah), `mariodf@uta.edu` (Mario Di Francesco), `giuseppe.anastasi@iet.unipi.it` (Giuseppe Anastasi), `mkumar@uta.edu` (Mohan Kumar)

1. Introduction

Wireless sensor networks (WSNs) have become an enabling technology for a wide range of applications [1]. Typically, a large number of sensor nodes are deployed over a geographical area. Sensors use multi-hop communication to send data acquired from the physical environment to a sink node or to an access point (AP) in the infrastructure. However, a dense WSN is not a requirement for many application scenarios, such as monitoring of weather conditions in large areas, air quality in urban scenarios, terrain conditions for precision agriculture, and so on. In this case, it is possible to exploit a *sparse wireless sensor network*, i.e., a WSN where the density of nodes is so low that they cannot communicate each other. In order to make communication feasible, data collection in sparse WSNs can be accomplished by means of *mobile data collectors* (MDCs) [2, 3]. MDCs are special mobile nodes responsible for data gathering and/or dissemination. They are assumed to be powerful in terms of data storage and processing capabilities, and are not energy constrained, in the sense that their energy source can be replaced or recharged easily. An MDC can serve either as a mobile sink (MS), a mobile node which is also the endpoint of data collection, or as a mobile relay (MR), which carries data from sensors to a sink node or an infrastructure AP. In either role, typically the MDC moves autonomously in the WSN [4].

Sparse WSNs with MDCs have many advantages over traditional dense WSNs [5]. First, costs are reduced, since fewer nodes can be deployed, as there is no need for a connected network. Second, as data is collected directly by the MDC from the sensor nodes, reliability is improved due to reduced congestion and collisions. Finally, data collection by MDC can extend the WSN lifetime, as the energy consumption is spread more uniformly in the network with respect to a static WSN, where the nodes close to the sink are usually more loaded than the others. However, data collection in sparse WSNs with MDCs also introduces several significant challenges, including energy-efficient MDC discovery and data collection.

A possible approach for energy-efficient data collection in sparse WSNs with MDCs requires that sensor nodes use a duty-cycle to discover presence of the MDCs in their communication range [4]. Moreover, the duty-cycle can be tuned according to the MDC arrivals, e.g., a sensor node can use a higher duty-cycle when there is a high probability that a MDC is in the communication range. This requires mechanisms to learn the arrival pattern of the MDC, even in the presence of uncertainty, and use that knowledge to adaptively tune the duty-cycle. To this end, solutions based on reinforcement learning [6] can be quite effective. In fact, reinforcement learning allows a sensor node to learn from the environment by merely interacting with it. Specifically, learning is based on the feedback from *tasks* (i.e., actions) performed by a node at any given state. A task can be selected either randomly or according to the accumulated knowledge, and the corresponding outcome is evaluated in terms of *reward*. A high reward means that the task is suitable to be executed in a given state, thus increasing the probability that the task will also be executed again in the future. As a consequence, reinforcement learning allows on-line learning and run-time adaptation by continuous interactions and feedback from the environment.

In this paper, we address the problem of data collection in sparse WSNs with MDCs, with focus on energy-efficient discovery of the mobile element. To this end, we propose the Resource-Aware Data Accumulation (RADA) framework, which exploits reinforcement learning to predict MDC arrivals and to adaptively tune a sensor node's duty-cycle for discovering the MDC. Our approach is quite simple, i.e., it demands minimal computational resources, and does not require any model of the environment. Therefore, it is very suitable for implementation on resource-constrained sensor nodes. RADA is based on discovery tasks with different duty-cycles, and on a state representation which is general enough to accommodate different MDC mobility patterns. As a consequence, RADA can be used in many sparse WSNs scenarios – including habitat monitoring and precision agriculture – without the need for application-specific strategies. We show through simulation experiments that the proposed solution can autonomously adapt to different application scenarios and diverse MDC mobility patterns, with a low energy-consumption and a high discovery efficiency.

The remainder of the paper is organized as follows. Section 2 overviews the related work, while Section 3 presents the reference network scenario and the considered mobility patterns. Section 4 describes the Distributed Independent Reinforcement Learning (DIRL) approach used for the design of RADA, which is presented in Section 5. Section 6 outlines the simulation setup, then Section 7 presents the experimental results. Finally, Section 8 concludes the paper.

2. Related work

Solutions for adaptive resource management and energy-efficient data collection in WSNs have already been considered in the literature. In the following, we provide an overview of the most relevant approaches for adaptive data collection, with particular focus on WSNs with mobile elements.

MDCs have been introduced first in opportunistic networks through the message ferrying approach [7]. In this context, a general framework for power management has been addressed by [8], and a knowledge-based approach to address the mobility pattern of the MDC has been proposed in [7]. However, as the proposed approach is devised for opportunistic networks, it cannot be used without being redesigned in the scenario considered in this paper. Many subsequent papers specifically focused on WSNs with MDCs, including [4, 9–12]. However, they assume that the operating parameters are chosen prior to deployment, and do not change with time. Clearly, these approaches lack flexibility, as they require an *a priori* characterization of some network parameters (e.g., the mobility pattern of the MDC, the duration of contacts or the message generation rate). In addition, the chosen parameters cannot adapt to changing operating conditions. An adaptive data collection scheme has been considered in [13]. However, it does not address MDC discovery, but assumes that some information on the MDC mobility pattern is available prior to deployment. In this work, instead, we provide an adaptive strategy which can be used even when there is limited knowledge on the mobility pattern of the MDC.

Knowledge-based approaches for data collection in WSNs with MDCs have been proposed in [14, 15]. In [14] the WSN is assumed to be rather dense, so that nodes can organize into clusters. Within each cluster, a specific node operates as a proxy, i.e., it collects data from other nodes in the cluster and relays them to the MDC. After detecting the presence of the MDC in their proximity, proxies initiate a reinforcement-based routing process so that messages are relayed to the destination while it traverses the network. Instead, [15] exploits reinforcement learning for discovery purposes, in the context of sparse WSNs where mobile nodes act as peers. Specifically, nodes scan for neighbors and use the number of encounters as a reward. The reward is mapped to a time-based domain. Then, sensor nodes perform discovery according to the likelihood of the other peers to be in contact, as per their energy budget. Although both [14] and [15] use reinforcement learning for data collection, they do not specifically address the problem of sparse WSNs with MDCs. In fact, the approach in [14] is more focused on routing, and assumes that the network is dense enough to form cluster of nodes. Even though the solution in [15] can also be applied to sparse WSNs, it has been specifically designed for sensor nodes acting as mobile peers. Instead, we consider WSNs where ordinary sensor nodes are static and only a limited number of special nodes (i.e., the MDCs) collect data in the network. In addition, we exploit an approach based on Q-learning [16], while the proposal in [15] is based on simple reinforcement learning. Finally, we design a solution which is flexible enough to support different mobility patterns in contrast to that in [15] which is optimized for multiple MDCs obeying certain schedules.

Adaptive data collection in WSNs has also been investigated by means of middleware solutions for proactive resource adaptation [17]. Among them, many solutions such as [18–20] actually focus on dense WSNs where nodes are static (or at most have a limited mobility), and assume some coordination between nodes which is difficult to achieve in sparse WSNs. To the best of our knowledge there are only a few solutions explicitly targeted to WSNs with mobile elements. Among them, Impala [21] is a middleware architecture proposed for optimizing the energy efficiency and reliability of WSN applications. However, Impala is targeted to scenarios where all nodes are mobile and act as peers with focus on application adaptation and update, rather than on resource allocation. Instead, the TinyLime [22] middleware has been proposed for the specific scenario of sparse WSNs. TinyLime provides mechanisms to perform data aggregation and tune the activity of nodes in order to save energy. However, the focus of TinyLime is on the proposed programming abstraction rather than on adaptation and resource management. In contrast, in this paper we propose an adaptive middleware approach to resource allocation for energy-efficient data collection in sparse WSNs.

In this paper, we extend the work in [23] by providing a more general solution which can be used for several different mobility patterns. Our scheme, which also supports additional features as well as multiple MDCs, is shown to be more energy-efficient than other approaches already proposed in the literature.

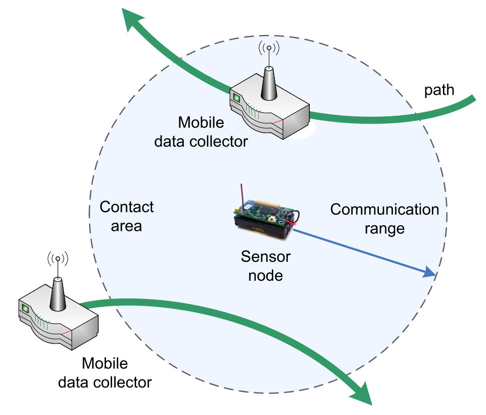
3. System Overview

In this section, we first describe the reference scenario and the different phases involved in data collection, along with the corresponding communication protocols. Next, we provide an overview of some significant MDC mobility models which will be later exploited as a reference for the design of our framework.

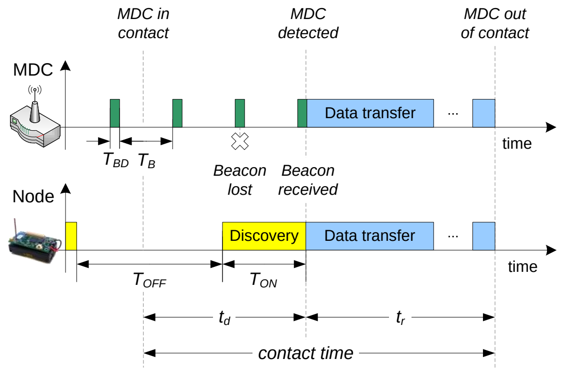
3.1. Reference network scenario

The reference network scenario is illustrated in Figure 1(a). Specifically, we assume that the network is sparse, i.e. at any time the MDC can communicate with at most one sensor node and vice versa (even when multiple MDCs are simultaneously present).

In the discussion below, we consider the communication between one MDC and an arbitrary static node in a sparse WSN. Data collection takes place only during a *contact*, i.e., when a sensor node and the MDC are within the communication range of each other. The area within the communication range of



(a)



(b)

Figure 1: Reference scenario (a) and an example of contact (b)

a sensor node is called *contact area*, and the overall time spent by the MDC within the contact area is called *contact time*. We also assume that the MDC mobility is not controllable, but has some periodicity. We further define as *tour* the smallest time interval after which the arrival pattern repeats [14] and *inter-contact time* as the actual time elapsed from the beginning of a contact to the beginning of the subsequent one.

Overall, the data collection process can be split into three main phases: *discovery*, *data transfer*, and *sleep* [8]. A sensor node enters the discovery phase for the timely detection of the MDC, as shown in Figure 1(b). Due to power management mechanisms and the unreliability of the communication channel, the successful MDC detection by a sensor node is not immediate, but rather delayed by a *discovery time* denoted by t_d. Upon detecting the MDC, the node switches from discovery state to data transfer state, and starts transmitting data to the MDC. Due to the necessity of a discovery process, a node cannot exploit the whole available contact time for data transfer. The portion of the contact time that can be actually used for subsequent data transfer is called *residual contact time* and is referred to as t_r. At the end of the data transfer phase, the node may switch to the discovery state again in order to detect the next MDC arrival. However, if the MDC has a (even partially) predictable mobility, a node can exploit this knowledge to further reduce its energy consumption [8]. In this case, the node can go to sleep until the next expected arrival of the MDC.

Similar to [4], we will assume an asynchronous discovery protocol. In detail, the MDC periodically sends special messages called *beacons* to advertise its presence in the surrounding area. The duration of a beacon message is given by T_{BD}, and subsequent beacons are spaced by a *beacon period* T_B. In order to save energy during the discovery phase, the node operates with a duty-cycle δ, whose active time $T_{ON} \geq T_B + T_{BD}$ so that a complete beacon can be received during the active time. This is needed since the active time of a sensor node is independent of the MDC presence in the contact area.

A node enters the data transfer phase upon receipt of a beacon from the MDC. While in this phase, the node remains always active to exploit the contact as much as possible. On the other hand, the MDC enters the data transfer phase as soon as it receives the first message sent by a sensor node, and stops transmitting beacons. Similar to [4], we assume that the communication protocol adopted for the data transfer phase is *selective repeat* [24], i.e., a window-based and an Automatic Repeat reQuest (ARQ) protocol with selective retransmission, whose window size is assumed to be equal to W messages. Note that acknowledgement messages in the ARQ scheme are used not only for implementing a retransmission strategy, but also as an indication of the MDC presence in the contact area [4]. The data transfer phase ends when either a sensor node has no more messages to transmit during a contact, or the MDC is not reachable any more. A node assumes that the MDC has exited the contact area when it misses N_{ack} consecutive acknowledgements.

3.2. Mobility patterns

The following discussion presents a number of mobility patterns as a reference for the design of the RADA framework that will be detailed in Section 5. The discussion also includes actual scenarios related to the considered mobility patterns.

- *Deterministic*: MDC arrivals are periodic while the inter-contact time is constant. Controlled MDCs such as robotized nodes [25] fall in this category.
- *Gaussian*: the MDC arrivals are periodic, but the inter-contact time follows a normal distribution. An example of such a mobility pattern is given by an uncontrolled MDC in an urban environment in the presence of traffic [13].
- *TimeOfDay*: MDC arrivals are still periodic, but the inter-contact time is generally variable. Specifically, MDC arrivals depend on a daily or weekly schedule as in [9]. Within the schedule, the actual MDC arrival can be either precise or not (e.g., similarly to the Gaussian mobility pattern).
- *TimeOfDay-Multiple*: this is similar to the previous one, but also accounts for multiple MDCs. Traffic and pollution monitoring applications exploiting either vehicles or people in high-density urban scenarios can be modeled with this mobility pattern. In this case, and especially when the number of MDCs is high, the sensor nodes should not rely on contacts with specific MDCs, but rather rely on the higher chance of any MDC passing through the contact area at a specific time of the day.

Different mobility patterns clearly impact on how the MDC arrivals can be learned, and on how efficiently they can be predicted. In the following, we design a framework, based on reinforcement learning [6], which can be easily adapted to the different mobility patterns, without the need for strategies specific to each scenario. In the next section, we will introduce the reinforcement learning strategy behind our resource-aware data collection framework, which will be detailed in Section 5.

4. Distributed Independent Reinforcement Learning (DIRL)

Distributed Independent Reinforcement Learning (DIRL) [26] is a framework for enabling autonomous and adaptive applications with inherent support for efficient resource management. The main idea of DIRL is to allow each individual sensor node to self-schedule its actions and allocate resources by learning the corresponding utility. At the same time, DIRL allows to meet application-defined constraints, as well as general goals (such as energy efficiency). Before describing DIRL in detail, in the following we present an overview of reinforcement learning.

Reinforcement learning (RL) is a branch of machine learning which is concerned with determining an optimum policy that maps states of the world to the actions that an agent should take (in the corresponding states) so as to maximize a payoff [6]. Specifically, the agent receives a numerical outcome (i.e., a *reward*), which provides a reinforcement signal, as a result of its own actions. Hence, the agent tries out different actions in order to learn what actions yield the highest reward. An action is selected either based on past experiences (*exploitation*) or randomly (*exploration*). As a consequence, RL is very useful for interactive (online) learning in dynamic and uncertain environments.

Let us first introduce the basic elements behind RL, and how they are mapped to the considered scenario. An *agent* is an entity which is able to sense the surrounding *environment* and perform some action. The actions performed by the agent consist of a number of *tasks*, which are scheduled according to the current *state*, i.e., a set of both application-defined and system variables which characterize the learning context. A *policy* determines how the agent selects a task according to the current state, while a *reward function* maps the state and the outcome of an action (i.e., task) to a numerical value. The learning process of the agent is guided by the reward function, with the purpose of maximizing the total reward with time. To this end, a *value function* is also used as indication of the long-term benefit from performing certain actions. The value function is built on top of the reward function, even though it abstracts from the immediate reward.

In our scenario, we consider each sensor node in the sparse WSN as an agent, and the combination of a sensor and one or more MDCs in the contact area as the environment. In addition, DIRL is based on independent learning, i.e., each agent applies the learning algorithm without any interaction with other agents. As a consequence, each agent can autonomously and dynamically self-configure in order to maximize its own reward. The main advantage of using independent learning in DIRL is that no coordination is required among sensor nodes, which is beneficial to scenarios where sensors are sparsely deployed and cannot easily communicate each other.

DIRL is based on Q-learning [16], a form of reinforcement learning which does not require a model of the environment (hence it is also called model-free). Q-learning uses a single data structure (i.e., an utility look-up table). The elements of the data structures are the utilities $Q(s,t)$ associated to the state s and the task t. In detail, the utility of performing a task t in a state s is defined as the expected value of sum of the immediate reward r and the discounted utility of resulting state s' after executing task t, i.e.

$$Q(s,t) = \mathbb{E}\left[r + \gamma \cdot e(s') | s, t\right] \tag{1}$$

where $e(s') = \max_t Q(s',t)$ over all tasks t. Note that the expected value above is conditioned to the state s and the task t. As Q-learning is done online, Equation (1) 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 equation:

$$Q(s,t) = (1-\alpha)Q(s,t) + \alpha\left[r + \gamma \cdot e(s')\right] \tag{2}$$

In Equation (2), α is a learning-rate parameter between 0 and 1, that 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 learnt utility value. Furthermore, γ is a discount-factor, also between 0 to 1; the higher the value, the greater the agent relies on future reward rather than on the immediate reward.

DIRL uses a weighted Hamming distance between two states in order to reduce the state space, which otherwise would be excessively large to be stored and managed by constrained sensor nodes. In addition to defining a state representation in the form of system and application variables, the application also specifies the weight associated to each variable. This weight specifies the significance of the corresponding variable in determining differences between two given states. Therefore, if an application state representation consists of the variables $V_1, V_2, \ldots, V_n$ with the corresponding weights $W_1, W_2, \ldots, W_n$, then DIRL uses the related information to determine if two given states s_1 and s_2 are similar or not. Specifically, it calculates the Hamming distance between the states as follows:

$$\begin{aligned} H(s_1 - s_2) &= W_1 \cdot |V_1(s_1) - V_1(s_2)| + W_2 \cdot |V_2(s_1) - V_2(s_2)| \\ &\quad + \ldots + W_n \cdot |V_n(s_1) - V_n(s_2)| \end{aligned} \tag{3}$$

Any two states s_j and s_k whose Hamming distance, as defined in Equation (3), is lower than a certain threshold θ – i.e., $H(s_j - s_k) < \theta$ – are considered similar, so that they can share a single entry in the Q data structure.

An important aspect of any RL system, including DIRL, 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 (e.g., starting with a high value and gradually decreasing to a low value). In this paper, the original DIRL exploration policy [26] has been improved by considering a mobility-aware exploration probability based

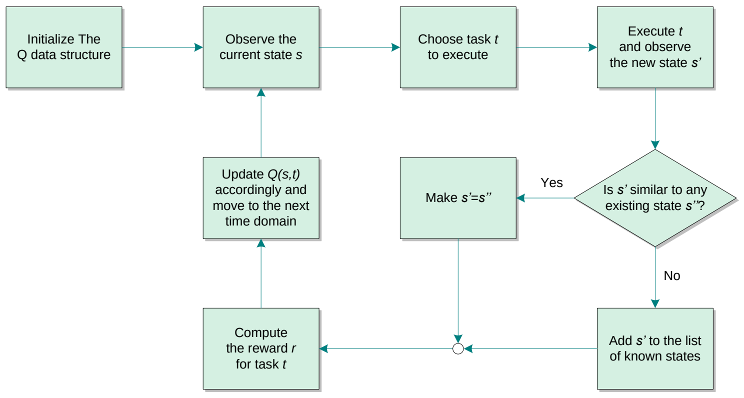

Initialize The Q data structure
Observe the current state s
Choose task t to execute
Execute t and observe the new state s'
Is s' similar to any existing state s''?
Yes
Make s'=s''
No
Add s' to the list of known states
Compute the reward r for task t
Update Q(s,t) accordingly and move to the next time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

Figure 2: Flowchart of the DIRL algorithm

on the number of contacts. More specifically, the exploration factor ϵ is given by

$$\epsilon = \epsilon_{min} + \max\left(0, (\epsilon_{max} - \epsilon_{min}) \cdot (c_{max} - c)/c_{max}\right) \tag{4}$$

where ϵ_{max} and ϵ_{min} define upper and lower bounds for the exploration factor, respectively; c represents the number of contacts detected by the agent at the time of evaluation, while c_{max} represents the maximum number of contacts for the learning policy. Note that c_{max} also controls the descending rate to the minimum exploration probability in Equation (4). In fact, the exploration probability is initially higher and gradually decreases over time towards ϵ_{min} as the agent is able to detect up to c_{max} contacts. Note that some minimum exploration is always required, so as to allow a sensor node to dynamically reconfigure in case of environmental changes.

DIRL needs the following inputs from the application: a set of tasks to be executed, in some priority order (which is important only until utilities are not established or if two tasks have same utility values); a reward function and an applicability predicate (incorporating application-specific constraints) associated with each task; a state representation consisting of both system and application variables, along with the corresponding weights for deriving the distance between states, and aggregating similar ones. Furthermore, it is necessary to define an appropriate task duration. To this end, we split the time – as perceived by the sensor – into a number of intervals called *time domains*, whose duration is denoted as T_d. More specifically, the sensor schedules a task for one time domain, then updates the utilities and evaluates the new state at the end of the same time domain.

After obtaining the input from the application, DIRL is executed according to the algorithm illustrated in Figure 2. Initially, all Q-values are set to zero. At each time domain DIRL selects a task to execute based on the exploration/exploitation strategy driven by Equation (4). Exploration selects an available task randomly, while exploitation selects the best task according to the learned utilities, i.e., the Q-values. After the execution of a task, DIRL observes the new state s' and compares it with all existing states based on a Hamming distance. If any existing state s'' has a Hamming distance to s' lower than θ, than s' is mapped to s''. Otherwise, a new state is created and added to the set of the current ones. Finally, DIRL computes the reward for the task t (executed while in the state s) and updates the corresponding Q-values accordingly.

5. Resource-Aware Data Accumulation (RADA)

In this section, we define an adaptive strategy based on DIRL for resource-aware data collection in sparse WSNs, namely, Resource-Aware Data Accumulation (RADA). The goal of this strategy is to maximize the number of contact detections, as well as the amount of data successfully transferred, while minimizing the energy consumption at the sensor nodes. In the following, we describe the task and state definitions (respectively) used by RADA as building blocks for a given data collection application.

5.1. Task definition

As discussed in Section 3.1, the main phases of the data collection process are represented by discovery, data transfer, and sleep. Since our goal is to learn the arrival pattern of the MDCs, we restrict our attention to discovery. Specifically, we have defined the tasks in RADA as discovery tasks with different duty-cycles. In order to make the derivation of tasks more general, we have defined the actual duty-cycles on the basis of a maximum allowed duty-cycle, denoted by δ_{max}, as follows.

- *High Duty-cycle* (HD). The sensor performs discovery with a high duty-cycle, equal to δ_{max}. Ideally, this task should be executed whenever there is a high probability that the MDC is in the contact area, so that the sensor node can discover it as early as possible.

- *Low Duty-cycle* (LD). The sensor performs discovery with a low duty-cycle, equal to $0.5 \cdot \delta_{max}$. Ideally this task should be executed whenever the probability of an MDC being in the contact area is low, so that the correspondent energy consumption is low as well (i.e., the sensor nodes spends most of the time sleeping).

- *Very Low Duty-cycle* (VLD). The sensor performs discovery with a very low duty-cycle, equal to $0.1 \cdot \delta_{max}$. Ideally this task should be executed whenever the probability of an MDC being in the contact area is very low, so that the correspondent energy consumption can be considered as almost negligible. However, discovery is still performed in order to adapt to eventual changes in the MDC mobility pattern.

Even though it is not a task itself, data transfer is performed whenever an MDC is detected, i.e., when a discovery task is successful (a beacon is successfully received by the sensor). In order to address this, we have introduced an internal variable i_c which is set to one when an MDC is considered to be in contact with the sensor. On the other hand, i_c is set to zero when the sensor has lost a number N_{ack} of consecutive acknowledgement messages as a result of the data transfer phase, thus assuming that the MDC has exited the contact area. Hence, the data transfer phase can be entered only after an MDC has been detected (i.e., $i_c = 1$).

Recall that the overall performance of data collection, especially in terms of energy consumption, depends on how efficiently MDC contacts are detected. Specifically, the efficiency of discovery can be maximized with a low energy-consumption if the tasks in RADA are scheduled according to the learned probability that the MDC is in contact (at a specific time). To this end, RADA learns this probability as the utility built by using the local rewards. Specifically, for all the tasks scheduled by a sensor node, the reward is defined as $r_t = (n_c \cdot m_p - 1) \cdot e_s$, where n_c is the number of contacts detected while executing that task, m_p is a price multiplier, and e_s the energy spent. Note that n_c can be greater than one, e.g., in the case where multiple MDCs enter the the contact area during the same time domain. The reason behind using a price multiplier is to allow a symmetric evaluation of the reward function. Thus, for each task, the reward is positive if at least one MDC is successfully detected. If no MDC is detected, the reward is negative (equal to minus e_s).

5.2. State definition

The state definition is one of the most important elements of schemes based on reinforcement learning. In fact, efficient learning heavily depends on how the state definition is suitable to characterize the environment. In general, different mobility patterns require different state definitions, since they have different requirements for learning their own context. For instance, the arrival of an MDC can be predicted based on the time elapsed between subsequent contacts (i.e., the inter-contact time) when arrivals are periodic. On the other hand, MDC arrivals can be predicted in terms of the hour of day (and/or of the week) when the MDC follows a certain schedule.

For the sake of flexibility, we provide a generic state definition whose variables are general enough to represent all the different mobility patterns presented in Section 3.2. Specifically, we define the following state variables.

- i_{ct}: the inter-contact time as observed by a sensor node.
- i_r: a boolean value (i.e., either 1 or 0) denoting whether (at least) an

MDC has been discovered or not.

- t_{od}: the time-of-day value, related to the specific time at which the state is evaluated.

Learning context can be customized to a specific scenario by appropriate tuning of the weights associated with different state variables. Clearly, more state variables can also be added if additional learning context is required. For instance, if the day of week is also important in addition to the time of the day, one more state variable representing the day of the week can be added. However, increasing the number of variables in the state representation also increases the state space and, as a consequence, the storage and computational requirements at the sensor node. Therefore, it is necessary to evaluate the effectiveness of additional variables in terms of overall performance before adding them.

As state variables are estimated by sensor nodes, we added some filtering techniques to avoid misinterpretation of context. For instance, a sensor might perceive a single actual MDC contact as multiple observed contacts. To this end, we implemented a simple timeout technique, so that the sensor considers a successful reception of a beacon message as a new contact only when a certain time has elapsed since the previous contact detection. Similarly, missing an actual contact would result in incorrect learning of the MDC arrival pattern. For this reason, we maintained a history of recent contacts (i.e., the related inter-contact time, where applicable), and used the minimum value of the sequence in order to cope with eventual missed contacts.

6. Simulation setup

To evaluate the performance of RADA, we used a custom discrete event simulator written in Java. In our analysis, we considered the following performance metrics.

- *Discovery ratio*: Ratio of the number of contacts correctly detected by the sensor to the total number of contacts. This metric characterizes the efficiency in discovering an MDC.

- *Residual contact ratio*: Ratio of the residual contact time to the total contact time. This metric is evaluated only for detected contacts, and represents how timely is the discovery. Specifically, the earlier the MDC is detected, the closer is the residual contact ratio to 100%, meaning that most of the contact time can be used for data transfer.

- *Activity ratio*: Ratio of the active time to the total time spent during discovery. Since it does not involve data transfer, this metric characterizes the average duty-cycle used for discovery.

- *Energy efficiency*: Average energy spent by a sensor for each contact correctly detected. It includes the energy spent for both discovery and data transfer, and it is obtained as the ratio of total energy consumed and the number of contacts detected.

As for the energy consumption, we used a simple model that characterizes the radio and does not consider the CPU, since the related energy consumption is almost negligible in most cases [1]. Specifically, the energy spent by the radio is calculated as $P_{state} \cdot T_{state}$, where P_{state} and T_{state} denote respectively, the power consumption of the radio and the amount of time spent in a given state, i.e., receive, transmit and sleep. We assume that the energy consumption of the radio during idle periods, i.e., when it is monitoring the channel, is the same as in the receive state.

In order to compare the performance of RADA with other approaches, we considered the following schemes.

- *Fixed High Duty-cycle* (FixedHD). The sensor node always executes the High Duty-cycle (HD) task. This scheme gives an upper bound on the performance achievable by schemes based on learning, at the cost of a high energy consumption.

- *Fixed Optimal Duty-cycle* (FixedOD). The sensor node only executes a task with a fixed duty-cycle, whose value is set to the average duty-cycle as obtained by RADA in the same operating conditions. As the duty-cycle

Table 1: Weights of the state variables used for simulation

Scenario(s)	**State variable**	**Weight**
Deterministic/Gaussian	Inter-contact time (i_{ct})	0.005
	MDC In Range (i_r)	1.0
	Time-of-day (t_{od})	0.0
Time-of-day	Inter-contact time (i_{ct})	0.0
	MDC In Range (i_r)	1.0
	Time-of-day (t_{od})	1.0
Multiple MDCs Time-of-day	Inter-contact time (i_{ct})	0.0
	MDC In Range (i_r)	0.0
	Time-of-day (t_{od})	1.0

is obtained dynamically by RADA, according to the actual scenario, this scheme is not feasible in practice and is used only for comparison purposes.

- *Oracle.* The sensor node has a perfect knowledge of MDC arrivals, hence it does not perform discovery at all. As a consequence, the sensor starts transmitting data as soon as the MDC enters the contact area, and stops whenever it does not have any more data or the MDC is out of contact. This is clearly an ideal scheme, and is used only as a reference.

As for the MDC, in our analysis we considered all the mobility patterns described in Section 3.2. The weights for the different state variables are set accordingly, as shown in Table 1. In detail, the weight of i_{ct} is set to a non-zero value only for the deterministic and Gaussian mobility patterns, which are thus learned by exploiting only the inter-contact time (since they are both periodic). On the other hand, the weight of t_{od} is set to a non-zero value only for the mobility patterns based on a schedule (i.e., both TimeOfDay and TimeOfDay-multiple), so that they can be learned according to the time of the day. Finally, the weight of i_r is set to a non-zero value only for the scenario with multiple MDCs, so that learning is not affected by the discovery of individual MDCs at a given time (since multiple independent MDCs can be present during the same

Table 2: Parameters used for simulation

Parameter	**Value**
Minimum exploration (ϵ_{min})	0.02
Maximum exploration (ϵ_{max})	0.3
State distance threshold (θ)	1.0
Maximum contacts (c_{max})	10
Time domain duration (T_d)	100 s
Maximum duty-cycle (δ_{max})	3.0
Price multiplier (m_p)	10
Beacon period (T_B)	100 ms
Beacon duration (T_{BD})	10 ms
Recent contacts history size	5
Standard deviation of the Gaussian mobility pattern	30 s
Message generation interval	10 s
Message payload size	24 bytes
Frame size	36 bytes
Window size (W)	16
Consecutive lost acknowledgements (N_{ack})	5
Radio transmit power (0 dBm)	49.5 mW
Radio receive/idle power	28.8 mW
Radio sleep power	0.6 μW

time domain).

In all experiments we performed 10 independent replicas, each consisting of at least 1000 MDC tours. We also derived confidence intervals with a 95% confidence level. In the following, we assume a MICA2 series mote [27] as the static sensor node, and use the related parameters for power consumption. We also assume that the radio is operating at a bitrate of 19.6 kbps. As for message losses, we used the model considered in [4, 13] and based on experimental data measured in a real testbed in the same scenario [28]. Specifically, the transmission range is 93 m, while the minimum distance between the MDC and the sensor node is 25 m. All other simulation parameters, chosen according to the methodology in [4], are summarized in Table 2.

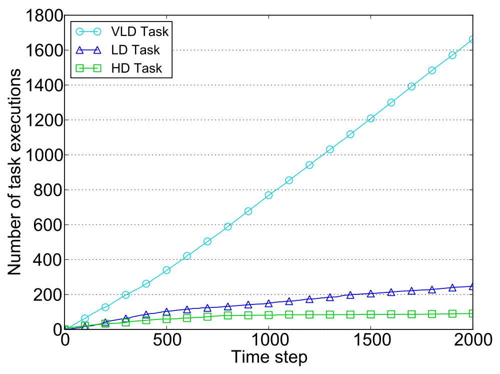
7. Experimental results

The evaluation of RADA is divided into different sections. First we focus on the performance of the learning algorithm, then we evaluate the impact of the time domain duration. Finally, we present results for different mobility patterns and speeds of the MDC. For the sake of clarity, in the following we will consider a single MDC which collects data from a single sensor node, unless stated otherwise. We also assume that the inter-contact time is 1800 s, with specific reference to a value reasonable for a tour by a vehicle in an urban scenario.

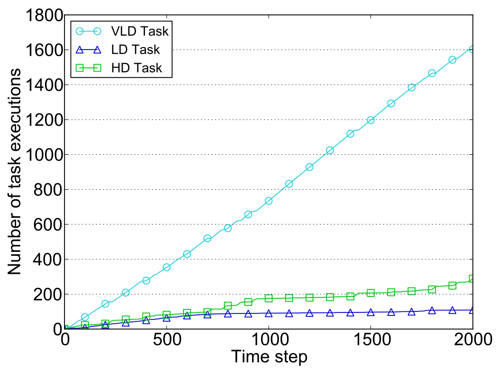
7.1. Performance of learning

In this section we evaluate the adaptive task scheduling capabilities of RADA. In the following, we use the deterministic mobility model and a fixed time domain duration of 100 s.

Figures 3(a) and 3(b) show the number of task executions as a function of time, when the MDC moves with a speed of 3.6 and 40 km/h, respectively. We can see that VLD (i.e., the task with the lowest duty-cycle) has the highest number of executions in both cases, and that the related slope is higher than the other two tasks. This happens since VLD consumes the least energy, and hence obtains the maximum reward when the MDC is not in the contact area. However, both HD and LD are still executed even after a steady-state is reached, i.e., when the exploration factor reaches its minimum value (i.e., after about 900 time domains). Specifically, Figure 3(a) shows that the slope of LD is greater than that of HD, meaning that the discovery task with a low duty-cycle is executed more than the discovery task with a high duty-cycle. This is consistent with the MDC mobility, which is rather low (i.e., 3.6 km/h), so that LD is adequate to discover the MDC. In contrast, Figure 3(b) shows the opposite trend, i.e., the slope of HD is greater than that of LD. This happens



(a)



(b)

Figure 3: Number of task executions over time for different MDC speeds: (a) 3.6 km/h and (b) 40 km/h

since the MDC has a high speed (i.e., 40 km/h) in the latter case, hence the low duty-cycle discovery task does not obtain enough reward when it is executed.

The different steady-state task execution patterns in the two cases clearly show that learning takes place, and that RADA adapts to the different mobility of the MDC. Specifically, the task with a higher duty-cycle is executed only when necessary, i.e., when the MDC speed is high, hence the probability of missing contacts is also high. As a result, RADA is able to preserve energy and also to efficiently schedule the different discovery tasks according to the operating conditions.

Table 3: Impact of the time domain duration on the energy consumption

Scenario	**Time domain duration** (T_d)	**Energy per contact** (mJ)
Deterministic	$i_{ct} \cdot 0.5\%$	1064.52
	$i_{ct} \cdot 5.0\%$	655.75
	$i_{ct} \cdot 25.0\%$	751.44
	Automatic (Initial value: $i_{ct} \cdot 0.5\%$)	647.93
	Automatic (Initial value: $i_{ct} \cdot 25.0\%$)	673.30
Gaussian	$i_{ct} \cdot 0.5\%$	1205.61
	$i_{ct} \cdot 5.0\%$	643.15
	$i_{ct} \cdot 25.0\%$	793.05
	Automatic (Initial value: $i_{ct} \cdot 0.5\%$)	630.06
	Automatic (Initial value: $i_{ct} \cdot 25.0\%$)	666.13

7.2. Impact of time domain duration

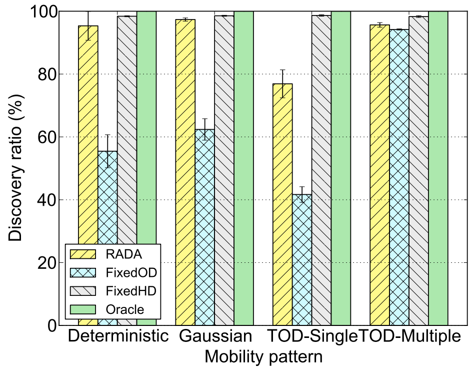
In this section, we evaluate the impact of the time domain duration (T_d) on performance, with specific reference to energy consumption. In fact, the time domain duration is a critical parameter, since each task is executed for the duration of a time domain, and then evaluated at the end of the related interval. As a consequence, if the time domain duration is not properly set, learning can be negatively affected. In order to evaluate the appropriate duration T_d of the time domain, we performed a set of preliminary experiments by considering both the deterministic and Gaussian mobility patterns, with a MDC speed of 20 km/h. In order to be general enough, we set the time domain duration as a fraction of the inter-contact time. The corresponding results are summarized in Table 3.

In the first set of experiments, we considered time domain durations equal to 0.5%, 5% and 25% of the inter-contact time i_{ct}, respectively. We can see that the lowest energy consumption is obtained when $T_d = i_{ct} \cdot 5\%$, for both the considered mobility patterns. The performance is worse when the time domain duration is either higher ($i_{ct} \cdot 25\%$) or lower ($i_{ct} \cdot 0.5\%$). This happens because of the difference between the time domain duration with respect to the contact time. In fact, if the time domain is much longer than the contact time, a task with higher duty-cycle can execute for an unnecessarily long time (e.g., either before or after the actual contact takes place). Similarly, a low value of the time domain duration can fragment the activity of the sensor, and even bias the learning process.

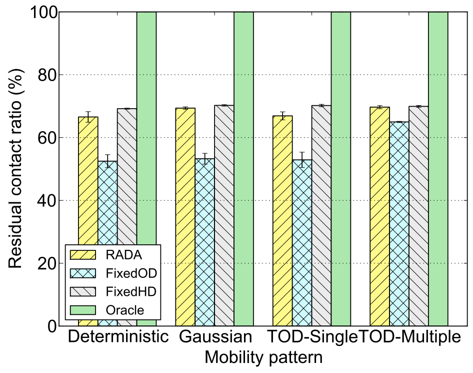
According to the results presented above, we implemented an automatic tuning mechanism which sets the time domain duration dynamically, based on the observed value of the inter-contact time. Specifically, the time domain duration is set to $T_d = i_{ct} \cdot 5\%$. However, an initial value of the time domain duration is needed to bootstrap the tuning process. In order to evaluate the impact of the initial time domain duration on the automatic tuning strategy, we performed an additional set of experiments. Specifically, the initial time domain duration for the automatic tuning strategy were set to the extreme cases of 0.5% and 25% of i_{ct}, respectively, as shown in Table 3. The results show that in both cases the automatic tuning strategy leads to a low energy consumption, which is close to that obtained with a fixed value of $T_d = i_{ct} \cdot 5\%$. Even though the energy consumption is a bit lower when the initial time domain duration is set to $i_{ct} \cdot 0.5\%$, both values are close to each other. As a consequence, the automatic tuning strategy is almost independent of the initial time domain duration, and also robust against variations in the operating conditions. Thus, in the following we use the automatic time domain tuning strategy for RADA.

7.3. Impact of mobility patterns

In this section, we evaluate the performance of RADA for all the mobility patterns introduced in Section 3.2. As already mentioned at the beginning of the section, we use an inter-contact time of 1800 s for both the Deterministic and Gaussian mobility patterns. For the TimeOfDay mobility pattern, the inter-contact time is varied according to the time of day. Specifically, the inter-contact time is set to 1800 s between 9 pm and 9 am, whereas it is set to 450 s during the rest of the day. We took a similar approach also for the TimeOfDay-Multiple mobility pattern, where the number of MDCs per hour is varied as a function



(a)

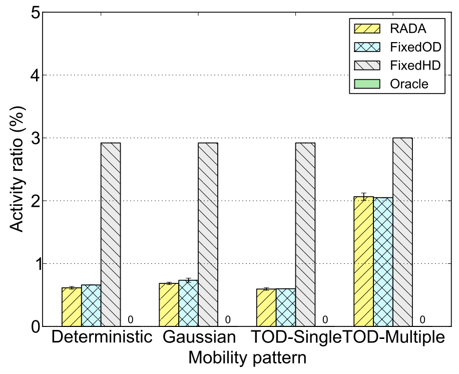


(b)

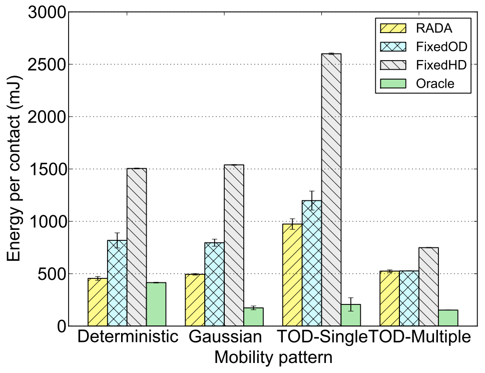
Figure 4: Discovery efficiency of the different schemes under different mobility patterns: (a) discovery ratio and (b) residual contact ratio

of the time of the day. Specifically, 20 MDCs are present between 9 am and 6 pm, 4 MDCs between 6 am and 9 am as well as between 6 pm and 9 pm, while no MDC visits the network between 9 pm and 6 am. In all cases, the MDC is moving at 20 km/h. In the following, we compare RADA against the other schemes described in Section 6.

The discovery ratio as a function of the mobility pattern is shown in Figure 4(a). We can clearly see that RADA has a very high discovery ratio, almost independent of the mobility pattern (it is only slightly lower for the TimeOfDay



(a)



(b)

Figure 5: Energy efficiency of the different schemes under different mobility patterns: (a) activity ratio and (b) energy consumption per contact

scenario), and very close to FixedHD and Oracle[1]. Furthermore, RADA shows significantly higher discovery ratio than FixedOD. Hence, RADA effectively learns the mobility pattern of the MDC. Indeed, the discovery ratio alone is not enough to characterize the effectiveness of the different schemes. In fact, the data transfer phase is significantly affected by the residual contact time. To this end, we also considered the residual contact ratio, illustrated in Figure

[1]Clearly, the Oracle scheme (which does not perform discovery at all) always obtains 100% discovery and residual contact ratios, and is shown here only as a reference.

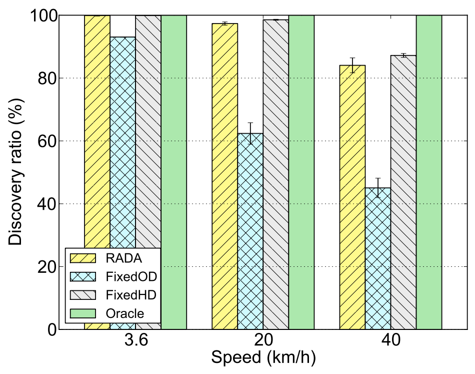
4(b), which quantifies the amount of the contact time that can be actually exploited by the data transfer phase (as a percentage). Here, we notice that RADA obtains a residual contact ratio always higher than FixedOD, and close to values achieved by FixedHD. As a consequence, the results clearly show that RADA can easily adapt to different mobility patterns.

We now focus on the activity ratio, shown in Figure 5(a), which characterizes the average duty-cycle used for discovering the MDC. RADA outperforms other schemes for the mobility patterns involving a single MDC, by achieving the lowest activity ratio (around 0.6%). Clearly, FixedOD has (almost) the same activity ratio as RADA by design, even though, as shown in Figure 4, it has a discovery efficiency which is much lower than that of RADA. The overall energy consumption per contact (hence including also data transfer) is shown in Figure 5(b). This metric jointly evaluates the discovery efficiency and the energy consumption, and summarizes the results of the previous analysis. From the figure, it is clear that RADA has the lowest energy consumption for all the mobility patterns, excluding Oracle (which is not feasible, however). RADA even achieves an energy consumption close to the Oracle scheme for the deterministic mobility pattern. In conclusion, sensor nodes can conserve energy due to RADA, as they can use a low duty-cycle while without significantly affecting the MDC detection efficiency.

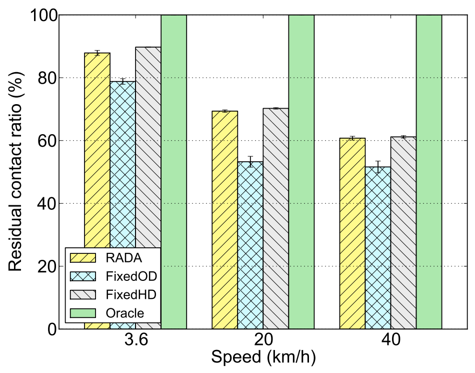
7.4. Impact of speed

In this section, we evaluate the impact of MDC speed on the performance of RADA. To this end, we assume that the MDC follows a Gaussian mobility pattern with a standard deviation of 30 s, and moves with a speed of 3.6, 20 and 40 km/h.

The discovery ratio as a function of the MDC speed is shown in Figure 6(a). We can see that all schemes obtain a discovery ratio close to 100% when the speed is low (i.e., 3.6 km/h). In addition, RADA has a discovery ratio higher than 80% even when the speed is high (i.e., 40 km/h), and close to FixedHD in any case. On the other hand, FixedOD is not very efficient even when the speed



(a)

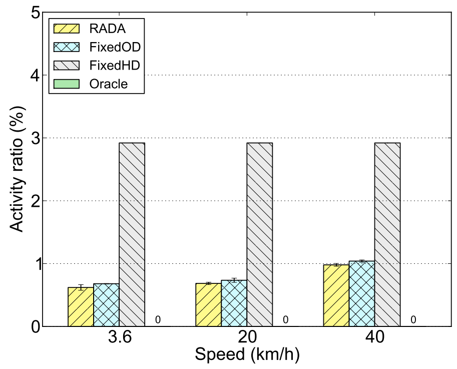


(b)

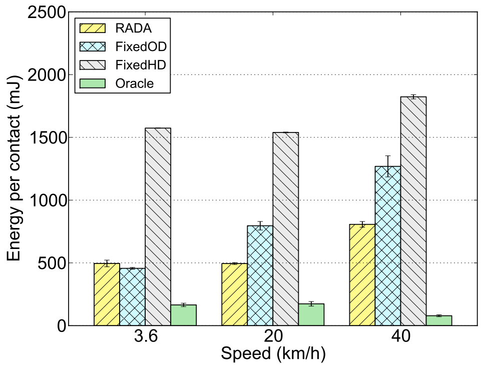
Figure 6: Discovery efficiency of the different schemes under different MDC speeds: (a) discovery ratio and (b) residual contact ratio

of the MDC is moderate or high. The impact of speed is even more apparent from the residual contact ratio, illustrated in Figure 6(b). In fact, almost all the different schemes can use a large share of the contact time when the speed is low. However, as the speed increases up to 20 km/h and above, the residual contact ratio reduces substantially. In any case, RADA has the highest residual contact ratio (obviously excluding Oracle), and is able to exploit more than 60% of the contact time even in the worst case.

The activity ratio is illustrated in Figure 7(a). Note that the activity ratio of RADA is consistently low (i.e., always lower than 1%) even though it slightly



(a)



(b)

Figure 7: Energy efficiency of the different schemes under different MDC speeds: (a) activity ratio and (b) energy consumption per contact

increases with the MDC speed. Clearly, FixedHD results in a much higher activity ratio, even three times higher than RADA. The low energy consumption of RADA is also testified by the energy spent per contact, as illustrated in Figure 7(b). In fact, RADA has the lowest energy expenditure among the (feasible) investigated schemes, and is not sensitive to the MDC speed. In summary, the evaluation showed that RADA is very efficient and robust, in the sense that it incurs a low energy consumption for a wide range of MDC mobility patterns, even when the contact time is short and the uncertainty on MDC arrivals is high.

8. Conclusions

In this paper we have proposed a novel Resource-Aware Data Accumulation (RADA) framework for sparse Wireless Sensor Networks (WSNs) with Mobile Data Collectors (MDCs). The issue of energy-efficient data collection has been addressed by exploiting a distributed reinforcement learning scheme, where each sensor node operates independent of the others. The proposed approach is specifically targeted to MDC discovery, and is based on a general state representation which is able to capture many realistic mobility patterns. Simulation results show that RADA is highly efficient, i.e., it reduces the average duty-cycle and the overall energy consumption of data collection, while detecting almost 100% of contacts. Compared to existing solutions, the proposed approach not only performs better, but also can adapt to different operating conditions and arrival patterns characterized by high uncertainty. As a result, RADA can be effectively employed in a wide range of applications in sparse WSNs with MDCs.

Acknowledgements

The research was funded partially by the US National Science Foundation awards CSR-0834493 and CNS-0721951. We are also grateful to the reviewers for their insightful comments and suggestions.

References

[1] G. Anastasi, M. Conti, M. Di Francesco, A. Passarella, Energy conservation in wireless sensor networks: A survey, Ad Hoc Networks 7 (3) (2009) 537–568.

[2] R. C. Shah, S. Roy, S. Jain, W. Brunette, Data mules: Modeling a three-tier architecture for sparse sensor networks, in: SNPA ‘03: Proceedings of the 1st IEEE Workshop on Sensor Network Protocols and Applications, 2003, pp. 30–41.

[3] D. Jea, A. Somasundra, M. Srivastava, Multiple controlled mobile elements (data mules) for data collection in sensor networks, in: DCOSS '05: Proceedings of the 1th IEEE International Conference on Distributed Computing in Sensor Systems, 2005, pp. 244–257.

[4] G. Anastasi, M. Conti, M. Di Francesco, Reliable and energy-efficient data collection in sparse sensor networks with mobile elements, Performance Evaluation 66 (12) (2009) 791–810.

[5] M. Di Francesco, S. K. Das, G. Anastasi, Data collection in wireless sensor networks with mobile elements: A survey, ACM Transactions on Sensor Networks 8 (1).

[6] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, 1st Edition, MIT Press, 1998.

[7] W. Zhao, M. Ammar, Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks, in: FTDCS '03: 9th IEEE International Workshop on Future Trends of Distributed Computing Systems, 2003, pp. 308–314.

[8] H. Jun, M. Ammar, E. Zegura, Power management in delay tolerant networks: A framework and knowledge-based mechanisms, in: SECON '05: Proceedings of the 2nd IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2005, pp. 418–429.

[9] A. Chakrabarti, A. Sabharwal, B. Aazhang, Using predictable observer mobility for power efficient design of sensor networks, in: IPSN '03: Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks, 2003, pp. 129–145.

[10] A. Kansal, A. Somasundara, D. Jea, M. Srivastava, D. Estrin, Intelligent fluid infrastructure for embedded networks, in: Mobisys '04: Proceedings of the 2nd ACM International Conference on Mobile Systems, Applications and Services, 2004, pp. 111–124.

[11] S. Jain, R. Shah, W. Brunette, G. Borriello, S. Roy, Exploiting mobility for energy efficient data collection in wireless sensor networks, ACM/Springer Mobile Networks and Applications 11 (3) (2006) 327–339.

[12] G. Anastasi, M. Conti, M. Di Francesco, An analytical study of reliable and energy-efficient data collection in sparse sensor networks with mobile elements, in: EWSN '09: Proceedings of the 6th European Conference on Wireless Sensor Networks, 2009, pp. 199–215.

[13] G. Anastasi, M. Conti, E. Monaldi, A. Passarella, An adaptive data-transfer protocol for sensor networks with Data Mules, in: WoWMoM '07: Proceedings of the 8th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2007, pp. 1–8.

[14] P. Baruah, R. Urgaonkar, B. Krishnamachari, Learning-enforced time domain routing to mobile sinks in wireless sensor fields, in: LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, 2004, pp. 525–532.

[15] V. Dyo, C. Mascolo, Efficient node discovery in mobile wireless sensor networks, in: DCOSS '08: Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems, 2008, pp. 478–485.

[16] R. Sutton, Temporal credit assignment in reinforcement learning, Ph.D. thesis, Department of Computer Science, University of Massachusetts, Amherst, MA 01003, published as COINS Technical Report 84-2 (1984).

[17] K. Henricksen, R. Robinson, A survey of middleware for sensor networks: state-of-the-art and future directions, in: Proceedings of the international workshop on Middleware for sensor networks, MidSens '06, 2006, pp. 60–65. doi:http://doi.acm.org/10.1145/1176866.1176877.

[18] W. Heinzelman, A. Murphy, H. Carvalho, M. Perillo, Middleware to support sensor network applications, IEEE Network 18 (1) (2004) 6–14.

[19] Y. Yu, B. Krishnamachari, , V. K. Prasanna, Issues in designing middleware for wireless sensor networks, IEEE Network 18 (2004) 15–21.

[20] P. J. Marron, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, K. Rothermel, TinyCubus: a flexible and adaptive framework sensor networks, in: EWSN ‘05: Proceedings of the 2nd European Workshop on Wireless Sensor Networks, 2005, pp. 278–289.

[21] T. Liu, M. Martonosi, 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, 2003, pp. 107–118.

[22] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, G. P. Picco, Mobile data collection in sensor networks: The TinyLime Middleware, Elsevier Pervasive and Mobile Computing Journal 4 (1) (2005) 446–469.

[23] M. Di Francesco, K. Shah, M. Kumar, G. Anastasi, An adaptive strategy for energy-efficient data collection in sparse wireless sensor networks, in: Proceedings of the 7th European Conference on Wireless Sensor Networks (EWSN 2010), 2010, pp. 322–337.

[24] J. F. Kurose, K. W. Ross, Computer Networking – A Top-Down Approach Featuring the Internet, 5th Edition, Addison-Wesley Professional, 2009.

[25] A. Somasundara, A. Kansal, D. Jea, D. Estrin, M. Srivastava, Controllably mobile infrastructure for low energy embedded networks, IEEE Transactions on Mobile Computing 5 (8) (2006) 1536–1233.

[26] K. Shah, M. Kumar, Distributed independent reinforcement learning (DIRL) approach to resource management in wireless sensor networks, MASS ‘07: Proceedings of the 4thIEEE Internatonal Conference on Mobile Adhoc and Sensor Systems) (2007) 1–9.

[27] Crossbow Technology, Mica2 wireless measurement system, http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.

[28] G. Anastasi, M. Conti, E. Gregori, C. Spagoni, G. Valente, Motes sensor networks in dynamic scenarios, International Journal of Ubiquitous Computing and Intelligence 1 (1).