SELF-ORGANIZING SYNCHRONICITY AND DESYNCHRONICITY USING REINFORCEMENT LEARNING

Mihail Mihaylov, Yann-Aël Le Borgne, Ann Nowé
Vrije Universiteit Brussel, Brussels, Belgium
{*mmihaylo,yleborgn,ann.nowe@vub.ac.be*}

Karl Tuyls
Maastricht University, Maastricht, The Netherlands
k.tuyls@maastrichtuniversity.nl

Keywords: reinforcement learning; synchronicity and desynchronicity; wireless sensor networks; wake-up scheduling;

Abstract: We present a self-organizing reinforcement learning (RL) approach for coordinating the wake-up cycles of nodes in a wireless sensor network in a decentralized manner. To the best of our knowledge we are the first to demonstrate how global synchronicity and desynchronicity can emerge through local interactions alone without the need of central mediator or any form of explicit coordination. We apply this RL approach to wireless sensor nodes arranged in different topologies and study how agents, starting with a random policy, are able to self-adapt their behavior based only on their interaction with neighboring nodes. Each agent independently learns to which nodes it should synchronize to improve message throughput and at the same with whom to desynchronize in order to reduce communication interference. The obtained results show how simple and computationally bounded sensor nodes are able to coordinate their wake-up cycles in a distributed way in order to improve the global system performance through (de)synchronicity.

1 INTRODUCTION

Synchronicity, which can be defined as the ability of a system to *organize simultaneous collective actions* (Werner-Allen et al., 2005), has long attracted the attention of biologists, mathematicians, and computer scientists. Seminal work on pulse-coupled oscillators of Mirollo and Strogatz (Mirollo and Strogatz, 1990) provided a theoretical framework for modeling the synchronization in biological systems such as fireflies flashings, heart pacemaker cells, or cricket chirps. The framework has recently been applied to systems of digital agents such as in the emerging field of wireless sensor networks (Knoester and McKinley, 2009), and to its counterpart, desynchronicity, in (Werner-Allen et al., 2005). In such multi-agent systems (MASs), synchronicity can be desirable to allow all agents to synchronously communicate, and desynchronicity to ensure that the communication channel is evenly shared among the agents.

Little attention has however been given to MASs where either global synchronicity *or* global desynchronicity of the system alone is impractical and/or undesirable. Desynchronicity refers to the state where the periodical activity of agents is shifted in time relative to one another, as opposed to synchronicity, where agents' activities coincide in time. In most MASs, an optimal solution is intuitively found where sets of agents synchronize with one another, but desynchronize with others. Nodes communicating in a wireless sensor network (WSN) are an example, as routing requires the synchronization of sets of nodes, while channel contention requires sets of nodes to desynchronize in order to avoid packet collisions. Other examples include traffic lights guiding vehicles through crossings in traffic control problems and jobs that have to be processed by the same machines at different times in job-scheduling problems. In such cases applying synchronization or desynchronization alone can be detrimental to the system (e.g. all traffic lights showing green, or complementary jobs processed at different time).

In these systems, agents should be logically organized in groups (or coalitions), such that the actions of agents within the group need to be synchronized, while *at the same time* being desynchronized with the actions of agents in other groups. We refer to this

concept for short as *(de)synchronicity* (for the system state), and *(de)synchronization* (for the process occurring at that state). An important characteristic of these systems is also that agents need to coordinate their actions without the need of centralized control.

In this paper, we show that coordinating the actions of agents (i.e., sensor nodes) can successfully be done using the reinforcement learning framework by rewarding successful interactions (e.g., transmission of a message in a sensor network) and penalizing the ones with a negative outcome (e.g., overhearing or packet collisions). This behavior drives nodes to repeat actions that result in positive feedback more often and to decrease the probability of unsuccessful interactions. Agents that tend to select the same successful action form a *coalition*.

A key feature of our approach is that no explicit notion of coalition is necessary. Rather, these coalitions emerge from the global objective of the system, and agents learn by themselves with whom they have to (de)synchronize (e.g. to maximize throughput in a routing problem). Here desynchronization refers to the situation where one agent's actions (e.g. waking up the radio transmitter of a wireless node) are shifted in time, relative to another, such that the (same) actions of both agents do not happen at the same time. For example, two traffic lights at a crossing are *desynchronized* when the light on one street shows green, while the other one displays red. In addition, the patterns of these traffic lights need to be *synchronized* with those on the nearby crossings to maximize traffic throughput. Thus, globally, one group of lights at consecutive crossings should be synchronized with each other and at the same time desynchronized with others to reach the global objective of high throughput taking into account the traffic flow.

We illustrate the benefits of combining synchronicity and desynchronicity in a scenario involving coordination in wireless sensor networks (WSNs). WSNs form a class of distributed and decentralized multi-agent systems, whose agents have very limited computational, communication and energy resources. A key challenge in these networks lie in the design of energy efficient coordination mechanisms, in order to maximize the lifetime of the applications. We apply our self-adapting RL approach in three wireless sensor networks of different topologies, namely line, mesh and grid. We show that nodes form coalitions which allow to reduce packet collisions and end-to-end latency, and thus to increase the network lifetime. This (de)synchronicity is achieved in a decentralized manner, without any explicit communication, and without any prior knowledge of the environment. To the best of our knowledge, we are the first to present a decentralized approach to coordination in WSNs *without any communication overhead* in the form of additional data required by the learning algorithm.

The paper is organized as follows. Section 2 guides the reader through some related work. It presents the background of our study and outlines the communication and routing protocols. The idea behind our approach is exposed in Section 3, together with its application in WSNs of three different topologies. The results of this work are analyzed in Section 4 shortly before we conclude in Section 5.

2 RELATED WORK

We proceed with the literature review on two fronts. We first present the challenging problem of (de)synchronization (also known as wake-up scheduling) in sensor networks, and then provide an overview of the synchronization (and desynchronization) mechanisms which have been proposed in the literature to solve the problem.

2.1 Wireless Sensor Networks: Coordination challenges

A Wireless Sensor Network is a collection of densely deployed autonomous devices, called *sensor nodes*, which gather data with the help of sensors (Ilyas and Mahgoub, 2005). The untethered nodes use radio communication to transmit sensor measurements to a terminal node, called the *sink*. The sink is the access point of the observer, who is able to process the distributed measurements and obtain useful information about the monitored environment. Sensor nodes communicate over a wireless medium, by using a multihop communication protocol that allows data packets to be forwarded by neighboring nodes to the sink.

When the WSN is deployed, the routing protocol requires that nodes determine their hop distance to the sink, i.e. the minimum number of nodes that will have to forward their packets. This is achieved by nodes broadcasting calibration packets only once immediately after deployment. Communication is done via a standard DATA/ACK protocol and messages are routed according to a shortest path algorithm with respect to the hop distance (Ilyas and Mahgoub, 2005). Nodes will start sending a packet at a random slot within the awake period to increase the chances of successful transmission for nodes that wake up at the same time.

Since communication is the most energy expensive action (Langendoen, 2008), it is clear that in order to save more energy, a node should turn off its

antenna (or go to sleep). However, when sleeping, the node is not able to send or receive any messages, therefore it increases the latency of the network, i.e., the time it takes for messages to reach the sink. High latency is undesirable in any real-time applications. On the other hand, a node does not need to listen to the channel when no messages are being sent, since it loses energy in vain. As a result, nodes should determine on their own *when* they should be awake within a frame. This behavior is called *wake-up scheduling*. Once a node wakes up, it remains active for a predefined amount of time, called *duty cycle*.

Wake-up scheduling in wireless sensor networks is an active research domain (Liang et al., 2007; Paruchuri et al., 2004; Liu and Elhanany, 2006; Cohen and Kapchits, 2009). A good survey on wake-up strategies in WSNs is presented in (Schurgers, 2007). In (Paruchuri et al., 2004) a randomized algorithm for asynchronous wake-up scheduling is presented that relies on densely deployed sensor nodes with means of localization. It also requires additional data to be piggybacked to messages in order to allow for making local decisions, based on other nodes. This bookkeeping of neighbors' schedules, however, introduces larger memory requirements and imposes significant communication overhead. A different asynchronous protocol for generating wake-up schedules (Zheng et al., 2003) is formulated as a block design problem with derived theoretical bounds. The authors derive theoretical bounds under different communication models and propose a neighbor discovery and schedule bookkeeping protocol operating on the optimal wake-up schedule derived. However, both protocols rely on localization means and incur communication overhead by embedding algorithm-specific data into packets. Adding such data to small packets will decrease both the throughput and the lifetime of the network.

A related approach that applies reinforcement learning in WSNs is presented in (Liu and Elhanany, 2006). As in the former two protocols, this approach requires nodes to include additional data in the packet header in order to measure the incoming traffic load. Moreover, the learning algorithm requires frame boundaries to be aligned in addition to slot boundaries.

2.2 Coordination and Cooperative Behavior

Coordination and cooperative behavior has recently been studied for digital organisms in (Knoester and McKinley, 2009), where it is demonstrated how populations of such organisms are able to evolve algorithms for synchronization based on biologically inspired models for synchronicity while using minimal information about their environment. Synchronization in WSNs, based on the Reachback Firefly Algorithm for synchronicity, is more specifically applied to WSNs in (Werner-Allen et al., 2005). The purpose of the study is to investigate the realistic radio effects of synchronicity in WSNs. Two complementary publications to the aforementioned work present the concept of desynchronicity in wireless sensor networks as the logical opposite of synchronicity (Degesys et al., 2007; Patel et al., 2007), where nodes perform their periodic tasks as far away as possible from all other nodes. Agents achieve that in a decentralized way by observing the firing messages of their neighbors and adjusting their phase accordingly, so that all firing messages are uniformly distributed in time.

The latter three works are based on the firefly-inspired mathematical model of pulse-coupled oscillators, introduced by Mirollo and Strogatz (Mirollo and Strogatz, 1990). In this seminal paper the authors proved that, using a simple oscillator adjustment function, any number of pulse-coupled oscillators would always converge to produce global synchronicity irrespective of the initial state. More recently, Lucarelli and Wang (Lucarelli and Wang, 2004) applied this concept in the field of WSNs by demonstrating that it also holds for multi-hop topologies.

The underlying assumption in all of the above work on coordination is that agents can observe each other's actions (or frequencies) and thus adapt their own policy (or phase), such that the system is driven to either full synchronicity or full desynchronicity, respectively. However, there are cases where agents are not able to observe the actions of others, thus rendering the above approaches inapplicable for such domains. For example sensor nodes could be in sleep mode while their neighbors wake up, thus they are unable to detect this event and adjust their wake-up schedule accordingly. Moreover, achieving either global synchronicity *or* global desynchronicity alone in most WSNs can be impractical or even detrimental to the system (see Section 1). In Section 3 we will present how we tackle these challenges using our decentralized reinforcement learning approach.

A related methodology is the collective intelligence framework (Wolpert and Tumer, 2008). It studies how to design large multi-agent systems, where selfish agents learn to optimize a private utility function, so that the performance of a global utility is increased. This framework, however, requires agents to store and propagate additional information, such as neighborhood's efficiency, in order to compute the world utility, to which they compare their own perfor-

mance. The approach therefore causes a communication overhead, which is detrimental to the network lifetime.

3 (DE)SYNCHRONICITY WITH REINFORCEMENT LEARNING

This section presents our decentralized approach to (de)synchronicity using the reinforcement learning framework. The proposed approach requires very few assumptions on the underlying networking protocols, which we discuss in Section 3.1. The subsequent sections detail the different components of the reinforcement learning mechanism.

3.1 Motivations and network model

All the previous approaches discussed in Section 2 require the agents to exchange information to achieve coordination. An important feature of our approach is that coordination is not explicitly *agreed* upon, but rather *emerges* as a result of the implicit communication generated by the data collection process.

Communication in WSNs is achieved by means of networking protocols, and in particular by means of the Medium Access Control (MAC) and the routing protocols (Ilyas and Mahgoub, 2005). The MAC protocol is the data communication protocol concerned with sharing the wireless transmission medium among the network nodes. The routing protocol allows to determine where sensor nodes have to transmit their data so that they eventually reach the sink. A vast amount of literature exists on these two topics (Ilyas and Mahgoub, 2005), and we sketch in the following the key requirements for the MAC and routing protocols so that our reinforcement learning mechanism presented in Section 3.2 can be implemented. We emphasize that these requirements are very loose.

We use a simple asynchronous MAC protocol that divides the time into small discrete units, called frames, where each frame is further divided into time slots. The frame and slot duration are application dependent and in our case they are fixed by the user prior to network deployment. The sensor nodes then rely on a standard duty cycle mechanism, in which the node is awake for a predetermined number of slots during each period. The awake period is fixed by the user, while the wake-up slot is initialized randomly for each node. These slots will be shifted as a result of the learning, which will coordinate nodes' wake-up schedules in order to ensure high data throughput and longer battery life. Each node will learn to be in active mode when its parents and children are awake, so that it forwards messages faster (synchronization), and stay asleep when neighboring nodes on the same hop are communicating, so that it avoids collisions and overhearing (desynchronization).

The routing protocol is not explicitly part of the learning algorithm and therefore any multi-hop routing scheme can be applied without losing the properties of our approach. In the experimental results, presented in Section 4, the routing is achieved using a standard shortest path multi-hop routing mechanism (cf. Section 2.1). The forwarding nodes need not be explicitly known, as long as they ensure that their distance to the sink is *lower* than the sender. Communication is done using a Carrier Sense Multiple Access (CSMA) protocol. Successful data reception is acknowledged with an ACK packet.We would like to note that the acknowledgment packet is necessary for the proper and reliable forwarding of messages. Our algorithm does use this packet to indicate a "correct reception" in order to formulate one of its reward signals (see Subsection 4.1). However, this signal is not crucial for the RL algorithm and thus the latter can easily function without acknowledgment packets. Subsection 3.3 will further elaborate on the use of reward signals.

It is noteworthy that the communication partners of a node (and thus the formation of coalitions) are influenced by the communication and routing protocols that are in use and not by our algorithm itself. These protocols only implicitly determine the *direction* of the message flow and not *who* will forward those messages, since nodes should find out the latter by themselves.

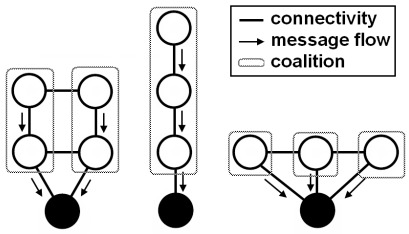


Figure 1: Examples of routing and coalition formation

Depending the routing protocol, coalitions (e.g., synchronized groups of nodes) logically emerge across the different hops, such that there is, if possible, only one agent from a certain hop within a coalition. Figure 1 illustrates this concept in three different topologies. It shows as an example how coalitions form as a result of the routing protocol. Intuitively, nodes from one coalition need to synchronize their wake-up schedules. As defined by the routing protocol, messages are not sent between nodes from the same hop, hence these nodes should desynchronize

(or belong to separate coalitions) to avoid communication interference. The emergence of coalitions will be experimentally illustrated for different topologies in Section 4.

3.2 Reinforcement learning approach: Methodology

Each agent in the WSN uses a reinforcement learning (RL) (Sutton and Barto, 1998) algorithm to learn an efficient wake-up schedule (i.e. when to remain active within the frame) that will improve throughput and lifetime in a distributed manner. It is clear that learning in multi-agent systems of this type requires careful exploration in order to make the action-values of agents converge. We use a value iteration approach similar to single-state Q-learning (Watkins, 1989) with an implicit exploration strategy, as subsection 3.5 will further elaborate on. However, our update scheme differs from that of traditional Q-learning (cf. subsection 3.4). The battery power required to run the algorithm is marginal to the communication costs and thus it is neglected. The main challenge in such a decentralized approach is to define a suitable reward function for the individual agents that will lead to an effective emergent behavior as a group. To tackle this challenge, we proceed with the definition of the basic components of the reinforcement learning algorithm as used by our methodology in the context of our application.

3.3 Actions and Rewards

The actions of each agent are restricted to selecting a time window (or a wake period) within a frame for staying awake. Since the size of these frames remains unchanged and they constantly repeat throughout the network lifetime, our agents use no notion of states, i.e. we say that our learning system is stateless. The duration of this wake period is defined by the duty cycle, fixed by the user of the system. In other words, each node selects a slot within the frame when its radio will be switched on for the duration of the duty cycle. Thus, the size of the action space of each agent is determined by the number of slots within a frame. In general, the more actions agents have, the slower the reinforcement learning algorithm will converge (Leng, 2008). On the other hand, a small action space might lead to suboptimal solutions and will impose an energy burden on the system. Setting the right amount of time slots within a frame requires a study on itself, that we shall not undertake in this paper due to space restrictions (see subsection 4.1 for exact values).

Every node stores a "quality value" (or Q-value) for each slot within its frame. This value for each slot indicates how beneficial it is for the node to stay awake during these slots for every frame, i.e. what is an efficient wake-up pattern, given its duty cycle and considering its communication history. When a communication event occurs at a node (overheard, sent or received a packet) or if no event occurred during the wake period (idle listening), that node updates the quality-value of the slot(s) when this event happened. The motivation behind this scheme is presented in subsection 3.5.

3.4 Updates and Action Selection

The slots of agents are initiated with Q-values drawn from a uniform random distribution between 0 and 1. Whenever events occur during node's active period, that node updates the quality values of the slots, at which the corresponding events occurred, using the following update rule:

$$Q_s^i \leftarrow (1-\alpha)\cdot \hat{Q}_s^i + \alpha \cdot r_{s,e}^i$$

where $Q_s^i \in [0,1]$ is the quality of slot s within the frame of agent i. Intuitively, a high Q_s^i value indicates that it is beneficial for agent i to stay awake during slot s. This quality value is updated using the previous Q-value ($\hat{Q}_s^i$) for that slot, the learning rate $\alpha \in [0,1]$, and the newly obtained reward $r_{s,e}^i \in [0,1]$ for the event e that (just) occurred in slot s. Thus, nodes will update as many Q-values as there are events during its active period. In other words, agent i will update the value Q_s^i for each slot s where an event e occurred. The latter update scheme differs from that of traditional Q-learning (Watkins, 1989), where only the Q-value of the selected action is updated. The motivation behind this update scheme is presented in subsection 3.5. In addition, we set here the future discount parameter γ to 0, since our agents are stateless (or single-state).

Nodes will stay awake for those consecutive time slots that have the highest sum of Q-values. Put differently, each agent selects the action $a_{s'}$ (i.e., wake up at slot s') that maximizes the sum of the Q-values for the D consecutive time slots, where D is the duty cycle, fixed by the user. Formally, agent i will wake up at slot s', where

$$s' = \arg\max_{s \in S} \sum_{j=0}^{D} Q_{s+j}^i$$

For example, if the required duty cycle of the nodes is set to 10% ($D = 10$ for a frame of $S = 100$ slots), each node will stay active for those 10 consecutive slots within its frame that have the highest sum of Q-values. Conversely, for all other slots the agent will

remain asleep, since its Q-values indicate that it is less beneficial to stay active during that time. Nodes will update the Q-value of each slot for which an event occurrs within its duty cycle. Thus, when forwarding messages to the sink, over time, nodes acquire sufficient information on "slot quality" to determine the best period within the frame to stay awake. This behaviour makes neighbouring nodes (de)synchronize their actions, resulting in faster message delivery and thus lower end-to-end latency.

3.5 Exploration

As explained in the above two subsections, active time slots are updated individually, regardless of when the node wakes up. The reason for this choice is threefold. Firstly, this allows each slot to be explored and updated more frequently. For example, slot s will be updated when the node wakes up anywhere between slots $s-1$ and $s-D+1$, i.e. in D out of S possible actions. Secondly, updating individual Q-values makes it possible to alter the duty cycle of nodes at run time (as suggest some preliminary results, not displayed in this paper) without invalidating the Q-values of slots. In contrast, if a Q-value was computed for each start slot s, i.e. the reward was accumulated over the wake duration and stored at slot s only, changing the duty cycle at run-time will render the computed Q-values useless, since the reward was accumulated over a different duration. In addition, slot s will be updated only when the agent wakes up at that slot. A separate exploration strategy is therefore required to ensure that this action is explored sufficiently. Thirdly, our exploration scheme will continuously explore and update not only the wake-up slot, but all slots within the awake period. Treating slots individually results in an implicit exploration scheme that requires no additional tuning.

Even though agents employ a greedy policy (selecting the action that gives the highest sum of Q-values), this "smooth" exploration strategy ensures that all slots are explored and updated regularly at the start of the application (since values are initiated randomly), until the sum of Q-values of one group of slots becomes strictly larger than the rest. In that case we say that the policy has converged and thus exploration has stopped. The speed of convergence is influenced by the duty cycle, fixed by the user, and the learning rate, which we empirically chose to be 0.1. A constant learning rate is in fact desirable in a non-stationary environment to ensure that policies will change with respect to the most recently received rewards (Sutton and Barto, 1998).

4 RESULTS

We proceed with the experimental comparison between our (de)synchronization approach and a fully synchronized one that is typically used in practice, such as S-MAC (Ye et al., 2004; Liu and Elhanany, 2006).

4.1 Experimental Setup

We applied our approach on three networks of different size and topology. In particular, we investigate two extreme cases where nodes are arranged in a 5-hop line (Figure 2(a)) and a 6-node single-hop mesh topology (Figure 3(a)). The former one requires nodes to synchronize in order to successfully forward messages to the sink. Intuitively, if any one node is awake while the others are asleep, that node would not be able to forwarded its messages to the sink. Conversely, in the mesh topology it is most beneficial for nodes to fully desynchronize to avoid communication interference with neighboring nodes. Moreover, the sink is able to communicate with only one node at a time. The third topology is a 4 by 4 grid (Figure 4(a)) where sensing agents need to both synchronize with some nodes and at the same time desynchronize with others to maximize throughput and network lifetime. The latter topology clearly illustrates the importance of combining synchronicity and desynchronicity, as neither one of the two behaviors alone achieves the global system objectives. Subsection 4.2 will confirm these claims and will elaborate on the obtained results.

Table 1: Parameters for experimental topologies

topology type	num. nodes	num. hops	avg. num. neighbors	data rate (msg/sec)
line	5	5	1.8	0.10
mesh	6	1	3.0	0.33
grid	16	4	3.25	0.05

Each network was ran for 3600 seconds in the OMNeT++ simulator (http://www.omnetpp.org/ – a C++ simulation library and framework,). Results were averaged over 50 runs with the same topologies, but with different data sampling times. Table 1 summarizes the differences between the experimental parameters of the three topologies.

Frames were divided in $S = 100$ slots of 10 milliseconds each, and we modeled five different events, namely overhearing ($r = 0$), idle listening ($r = 0$ for each idle slot), successful transmission ($r = 1$ if ACK received), unsuccessful transmission ($r = 0$ if no ACK

received) and successful reception ($r = 1$). Maximizing the throughput requires both proper transmission as well as proper reception. Therefore, we treat the two corresponding rewards equally. Furthermore, most radio chips require nearly the same energy for sending, receiving (or overhearing) and (idle) listening (Langendoen, 2008), making the three rewards equal. We consider these five events to be the most energy expensive or latency crucial in wireless communication. Additional events were also modeled, but they were either statistically insignificant (such as busy channel) or already covered (such as unsuccessful transmissions instead of collisions).

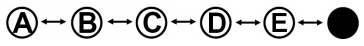
Due to the exponential smoothing nature of the reward update function (cf. subsection 3.4) the Q-values of slots will be shifted towards the latest reward they receive. We would expect that the "goodness" of slots will decrease for negative events, and will increase for successful communication. Therefore, the feedback agents receive is binary, i.e. $r^i_{s,e} \in \{0,1\}$, since it carries the necessary information. Other reward signals were also evaluated, resulting in similar performance.

To better illustrate the importance and effect of (de)synchronicity in these topologies, we compare our approach to networks where all nodes wake up at the same time. All other components of the compared networks, such as the routing and CSMA communication protocols, remain the same. In other words, we compare our RL technique to networks with no coordination mechanism, but which employ some means of time synchronization, the small overhead of which will be neglected for the sake of a clearer exposition. This synchronized approach ensures high network throughput and is already used by a number of state-of-the-art MAC protocols, such as RL-MAC (Liu and Elhanany, 2006) and S-MAC (Ye et al., 2004). However, as we will demonstrate in subsection 4.2, synchronization alone will in fact decrease system performance.

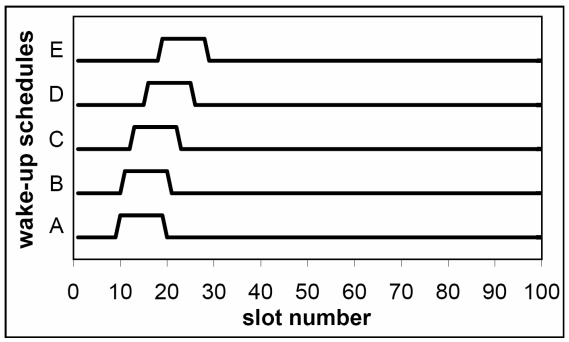
4.2 Evaluation

Figure 2(b) displays the resulting schedule of the line topology (Figure 2(a)) after the action of each agent converges. The results indicate that nodes have successfully learned to stay awake at the same time in order for messages to be properly forwarded to the sink. In other words, we observe that all nodes belong to the same coalition, as suggested in Figure 1. If any one node in the line topology had remained active during the sleep period of others, its messages, together with those of its higher hop neighbors would not have been delivered to the sink. Even though neighboring nodes are awake at the same time (or have synchronized), one can see that schedules are slightly shifted in time. The reason for this desynchronicity is to reduce overhearing of messages from lower hop nodes and to increase throughput – a behavior that nodes have learned by themselves. The size of this time shift depends on the time at which nodes send data within their awake period. As mentioned in subsection 2.1, messages are sent at a random slot within a frame. Therefore, the difference between the wake-up times is small enough to increase the chance of successful transmissions and large enough to ensure fast throughput, compensating for propagation delays.

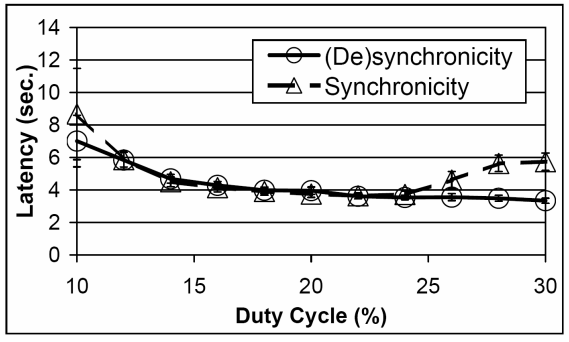
In the line topology this time shift is, however, marginal to the active period and therefore the performance of the learning nodes is comparable to that of the fully synchronized network. This effect can be observed in Figure 2(c), which displays the end-to-end latency of the learning and the synchronized nodes respectively. We can conclude from the graph that in a line topology, where nodes have no neighbors on the same hop, the improvements of our learning



(a) Line topology



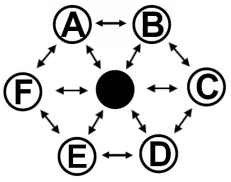
(b) Resulting wake-up schedule after convergence for duty cycle of 10%



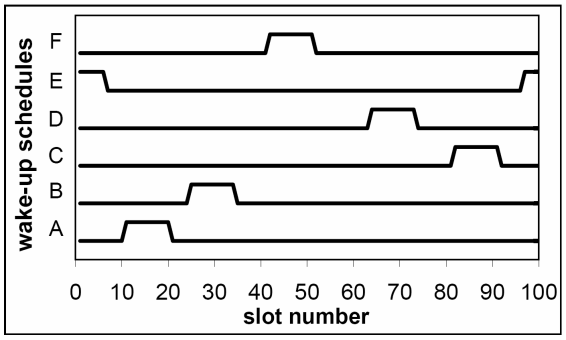
(c) End-to-end latency for different duty cycles

Figure 2: Experimental results for the line topology

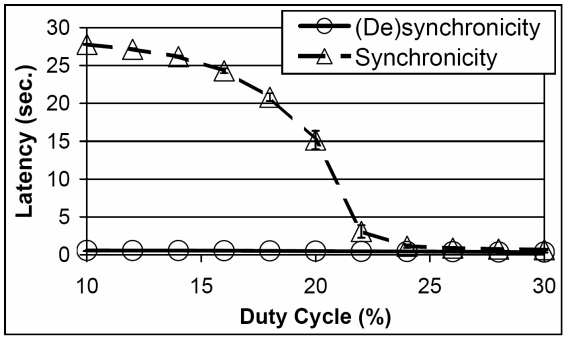
algorithm are marginal to that of a synchronized network. As mentioned above, the reason for this comparable performance lies in the fact that a successful message forwarding in the line topology requires synchronicity. Nevertheless, our agents are able to independently achieve this behavior without any communication overhead.



(a) Mesh topology



(b) Resulting wake-up schedule after convergence for duty cycle of 10%



(c) End-to-end latency for different duty cycles

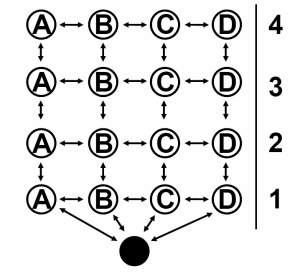
Figure 3: Experimental results for the mesh topology

In contrast to the previous topology, our second set of experiments investigate the performance of the network where all nodes lie on the same hop from the sink. This setup presents agents with the opposite challenge, namely to find an active period where no other node is awake. The latter behavior will eliminate communication interference with neighboring nodes and will ensure proper reception of messages at the sink. Figure 3(b) displays the wake-up schedule of the learning nodes for a duty cycle of 10% after the actions of agents converge. One can observe that the state of desynchronicity has been successfully achieved where each node is active at a different time within a frame. Put differently, each node has chosen a different wake-up slot and therefore belongs to different coalition. The benefit of this desynchronized pattern is clearly evident in Figure 3(c) where we compare it to the latency of the synchronized system. For duty cycles lower than 25% the reduction in latency is signifficant, as compared to a synchronized network of the same topology.

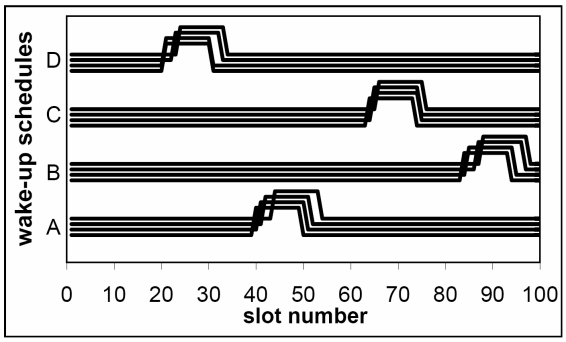
Lastly, we investigate a "mix" of the above two topologies, namely a grid shown in Figure 4(a). Nodes here need to synchronize with those that lie on the same branch of the routing tree to ensure high throughput, while at the same time desynchronize with neighboring branches to avoid communication interference. The resulting schedule of the learning nodes after convergence is displayed in Figure 4(b). As expected, the four columns of nodes belong to four different coalitions, where nodes in one coalition are synchronized with each other (being active nearly at the same time) and desynchronized with the other coalitions (sleeping while others are active). This is the state of (de)synchronicity. Nodes in one coalition exhibit comparable behavior to those in a line topology, i.e. they have synchronized with each other, while still slightly shifted in time. At the same time nodes on one hop have learned to desynchronize their active times similar to the mesh topology.

The result of applying our learning approach in a grid topology for various duty cycles can be observed in Figure 4(c). It displays the average end-to-end latency of the network when using synchronicity and (de)synchronicity respectively. For duty cycles lower than 20% the improvements of our learning nodes over a synchronized network are outstanding. Due to the high data rate, the synchronized nodes are incapable of delivering all packets within the short awake period, resulting in increased latency and high packet loss. The high amount of dropped packets at low duty cycles is simply intolerable for real-time applications. This reduced performance at low duty cycles is due to the large number of collisions and re-transmissions necessary when all nodes wake up at the same time. The learning approach on the other hand drives nodes to coordinate their wake-up cycles and shift them in time, such that nodes at neighboring coalitions desynchronize their awake periods. In doing so, nodes effectively avoid collisions and overhearing, leading to lower end-to-end latency and longer network lifetime.

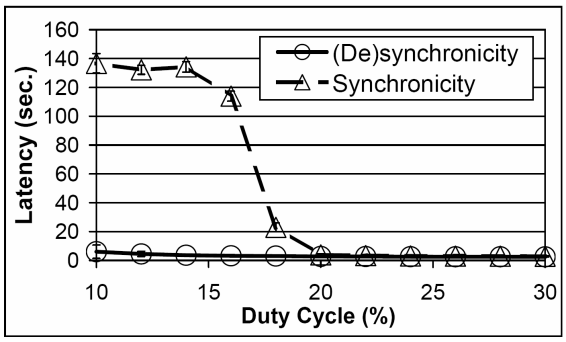
When nodes coordinate their actions, they effectively reduce communication interference with neighboring nodes. This behavior results in lower amount



(a) Grid topology



(b) Resulting wake-up schedule after convergence for duty cycle of 10%



(c) End-to-end latency for different duty cycles

Figure 4: Experimental results for the grid topology

of overheard packets, less collisions and therefore fewer retries to forward a message, as compared to the fully synchronized network.

Finally, we would like to mention the convergence rate of the learning agents. The implicit exploration scheme, described in subsection 3.5 makes nodes select different actions in the beginning of the simulation in order to determine their quality. As time progresses, the Q-values of slots are updated sufficiently enough to make the policy of the agents converge. We measured that after 500 iterations on average the actions of agents do not change and thus the state of (de)synchronicity has been reached. In other words, after 500 seconds each node finds the wake-up schedule that improves message throughput and minimizes communication interference. This duration is sufficiently small compared to the lifetime of the system for a static WSN, which is in the order of several days up to a couple of years depending on the duty cycle and the hardware characteristics (Ilyas and Mahgoub, 2005).

4.3 Discussion

The main feature of the proposed reinforcement learning approach is that nodes learn to cooperate by simply observing their own reward signals while forwarding messages. As a result, it drives agents to coordinate their wake-up cycles without any communication overhead. This behavior ensures that neighboring nodes avoid communication interference without exchanging additional data. The synchronized approach, on the other hand, lets all nodes be awake at the same time. Thus, for low duty cycles, the risks of collisions and therefore re-transmissions are increased. This was confirmed by our experimental results, where we observed that (de)synchronicity can increase the system performance in mesh and grid topologies for low duty cycles as compared to a standard, fully synchronized approach.

When duty cycles of (synchronized) agents become larger, nodes have less chance of collisions and hence re-transmissions, leading to decreased latency and no packet loss. As mentioned in subsection 2.1, packets are sent at random slots within the active period. Thus, the negative effect of being awake at the same time becomes less pronounced as the duty cycle increases. Similarly, as the number of messages per active period decreases, the learning agents receive less reward signals, leading to slower convergence and poorer adaptive behavior. In such cases it is less beneficial to apply our RL approach, as it performs similar to the fully synchronized one. Nevertheless, synchronized nodes still overhear packets of neighbors, resulting in higher battery consumption as compared to nodes that use our learning algorithm.

Different axes may finally be considered to improve the efficiency of the proposed (de)synchronization policy. First, we only considered fixed duty-cycles, whereas the traffic in WSN is often unequal. In particular, nodes closer to the sink often undergo higher traffic loads due to data forwarding. A possible approach would be to adopt a *cross-layering* strategy where information from the routing layer, such as the node depth or the expected traffic load, would be used to adapt the duty cycle. Second, the RL approach is not well suited for irregular data traffic, as such type of traffic is likely

to impair the convergence rate of the RL procedure. A related issue concerns the clock drifts, which may also cause the learning procedure to fail to converge to a stable scheduling. We plan to address these issues in our future work.

5 CONCLUSIONS

In this paper we presented a decentralized reinforcement learning (RL) approach for self-organizing wake-up scheduling in wireless sensor networks (WSNs). Our approach drives nodes to coordinate their wake-up cycles based only on local interactions. In doing so, agents independently learn both to synchronize their active periods with some nodes, so that message throughput is improved, and at the same time to desynchronize with others in order to reduce communication interference. We refer to this concept as (de)synchronicity. We investigated three different topologies and showed that agents are able to independently adapt their duty cycles to the routing tree of the network. For high data rates this adaptive behavior improves both the throughput and lifetime of the system, as compared to a fully synchronized approach where all nodes wake up at the same time. We demonstrated how initially randomized wake-up schedules successfully converge to the state of (de)synchronicity without any form of explicit coordination. As a result, our approach makes it possible that agent coordination *emerges* rather than is *agreed* upon.

ACKNOWLEDGMENTS

The authors of this paper would like to thank the anonymous reviewers for their useful comments and valuable suggestions. This research is funded by the agency for Innovation by Science and Technology (IWT), project IWT60837; and by the Research Foundation - Flanders (FWO), project G.0219.09N.

REFERENCES

Cohen, R. and Kapchits, B. (2009). An optimal wake-up scheduling algorithm for minimizing energy consumption while limiting maximum delay in a mesh sensor network. *IEEE/ACM Trans. Netw.*, 17(2):570–581.

Degesys, J., Rose, I., Patel, A., and Nagpal, R. (2007). Desync: self-organizing desynchronization and tdma on wireless sensor networks. In *Proceedings of the 6th IPSN*, pages 11–20, New York, NY, USA. ACM.

http://www.omnetpp.org/ – a C++ simulation library and framework.

Ilyas, M. and Mahgoub, I. (2005). *Handbook of sensor networks: compact wireless and wired sensing systems*. CRC.

Knoester, D. B. and McKinley, P. K. (2009). Evolving virtual fireflies. In *Proceedings of the 10th ECAL*, Budapest, Hungary.

Langendoen, K. (2008). Medium access control in wireless sensor networks. *Medium access control in wireless networks*, 2:535–560.

Leng, J. (2008). *Reinforcement learning and convergence analysis with applications to agent-based systems*. PhD thesis, University of South Australia.

Liang, S., Tang, Y., and Zhu, Q. (2007). Passive wake-up scheme for wireless sensor networks. In *Proceedings of the 2nd ICICIC*, page 507, Washington, DC, USA. IEEE Computer Society.

Liu, Z. and Elhanany, I. (2006). Rl-mac: a reinforcement learning based mac protocol for wireless sensor networks. *Int. J. Sen. Netw.*, 1(3/4):117–124.

Lucarelli, D. and Wang, I.-J. (2004). Decentralized synchronization protocols with nearest neighbor communication. In *Proceedings of the 2nd SenSys*, pages 62–68, New York, NY, USA. ACM.

Mirollo, R. E. and Strogatz, S. H. (1990). Synchronization of pulse-coupled biological oscillators. *SIAM J. Appl. Math.*, 50(6):1645–1662.

Paruchuri, V., Basavaraju, S., Durresi, A., Kannan, R., and Iyengar, S. S. (2004). Random asynchronous wakeup protocol for sensor networks. In *Proceedings of the 1st BROADNETS*, pages 710–717, Washington, DC, USA. IEEE Computer Society.

Patel, A., Degesys, J., and Nagpal, R. (2007). Desynchronization: The theory of self-organizing algorithms for round-robin scheduling. In *Proceedings of the 1st SASO*, pages 87–96, Washington, DC, USA. IEEE Computer Society.

Schurgers, C. (2007). *Wireless Sensor Networks and Applications*, chapter Wakeup Strategies in Wireless Sensor Networks, page 26. Springer.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Watkins, C. (1989). *Learning from delayed rewards*. PhD thesis, University of Cambridge, England.

Werner-Allen, G., Tewari, G., Patel, A., Welsh, M., and Nagpal, R. (2005). Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of the 3rd SenSys*, pages 142–153, New York, NY, USA. ACM.

Wolpert, D. H. and Tumer, K. (2008). An introduction to collective intelligence. Technical Report NASA-ARC-IC-99-63, NASA Ames Research Center.

Ye, W., Heidemann, J., and Estrin, D. (2004). Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 12(3):493–506.

Zheng, R., Hou, J. C., and Sha, L. (2003). Asynchronous wakeup for ad hoc networks. In *Proceedings of the 4th MobiHoc*, pages 35–45, New York, NY, USA. ACM.