Reinforcement Learning for Self-Organizing Wake-Up Scheduling in Wireless Sensor Networks

Mihail Mihaylov[1], Yann-Aël Le Borgne[1], Karl Tuyls[2], and Ann Nowé[1]

[1]Vrije Universiteit Brussel, Brussels, Belgium
{mmihaylo,yleborgn,ann.nowe}@vub.ac.be
[2]Maastricht University, Maastricht, The Netherlands
k.tuyls@maastrichtuniversity.nl

Abstract. Wake-up scheduling is a challenging problem in wireless sensor networks. It was recently shown that a promising approach for solving this problem is to rely on reinforcement learning (RL). The RL approach is particularly attractive since it allows the sensor nodes to coordinate through local interactions alone, without the need of central mediator or any form of explicit coordination. This article extends previous work by experimentally studying the behavior of RL wake-up scheduling on a set of three different network topologies, namely line, mesh and grid topologies. The experiments are run using OMNET++, a the state-of-the-art network simulator. 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. The main insight of these experiments is to show that sensor nodes learn to synchronize if they have to cooperate for forwarding data, and learn to desynchronize in order to avoid interferences. This synchronization/desynchronization behavior, referred to for short as (de)synchronicity, allows to improve the message throughput even for very low duty cycles.

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

1 INTRODUCTION

A Wireless Sensor Network is a collection of densely deployed autonomous devices, called *sensor nodes*, which gather data with the help of sensors [4]. 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 multi-hop communication protocol that allows data packets to be forwarded by neighboring nodes to the sink. A typical multi-hop communication protocol is to rely on a shortest path tree with respect to the

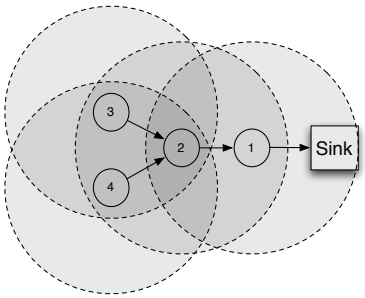


Fig. 1. Sensor nodes connected to a base station by means of a multi-hop routing tree. Grayed circles indicate overlapping communication regions.

hop distance [4]. Such a tree is obtained by letting nodes broadcast packets after deployment, in order identify their neighbors. The nodes then determine the neighbor node which is the closest (in terms of hops) to the sink, and use it as the relaying node for the multi-hop routing. An example of multi-hop shortest path routing structure is given in Fig. 1, together with the radio communication ranges of sensor nodes.

Since communication is the most energy expensive action, it is clear that in order to save energy, a node should turn off its antenna (or go to sleep) [5]. 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 [7, 12, 8, 2]. A good survey on wake-up strategies in WSNs is presented in [14]. The standard approach is S-MAC, a synchronized medium access control (MAC) protocol for WSN [17]. In S-MAC, the duty-cycle is fixed by the user, and all sensor nodes synchronize in such a way that their active periods take place at the same time. This synchronized active period enables neighboring nodes to communicate with one another. The use of routing then allows any pair of nodes to exchange messages. By tuning the duty-cycle, wake-up scheduling therefore allows to adapt the use of sensor resources to the application requirements in terms of latency, data rate and lifetime [14].

Recently, we showed that the wake-up scheduling problem could be efficiently tackled in the framework of multi-agent systems and reinforcement learning. In wireless sensor networks, the sensor nodes can be seen as agents, which have to logically self-organize in groups (or *coalitions*). The actions of agents within a group need to be synchronized (e.g., for data forwarding), while *at the same time*

being desynchronized with the actions of agents in other groups (e.g., to avoid radio interferences). We refer to this concept for short as *(de)synchronicity*.

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) [10]. This behavior drives the nodes to repeat actions that result in positive feedback more often and to decrease the probability of unsuccessful interactions. *Coalitions* are formed when agents select the same successful actions. 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.

In this article, we extend our previous results by illustrating the benefits of 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, even for very low duty cycles. This (de)synchronicity is achieved in a decentralized manner, without any explicit communication, and without any prior knowledge of the environment. Our simulations are implemented using OMNET++, a state-of-the-art simulator [11].

The paper is organized as follows. Section 2 presents the reinforcement learning approach for solving the wake-up scheduling problem in WSN. Section 3 analyzes and discusses the performances of the RL approach on three different topologies, namely line, mesh and grid topologies. We also compare it to the standard S-MAC protocol and briefly discusses future work. Section 4 concludes this paper.

2 (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 2.1. The subsequent sections detail the different components of the reinforcement learning mechanism.

2.1 Motivations and network model

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 [4]. 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 [4], and we sketch in the following the key requirements for the MAC and routing protocols so that our reinforcement learning mechanism presented in Section 2.2 can be implemented. We emphasize that these requirements are very loose.

We use a simple MAC protocol, inspired from S-MAC [17], that divides the time into small discrete units, called frames. We further divide each frame 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 duration of the awake period is fixed by the user, while its position is initialized randomly within the frame for each node. These active 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 3, the routing is achieved using a standard shortest path multi-hop routing mechanism. 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 3.1). However, this signal is not crucial for the RL algorithm and thus the latter can easily function without acknowledgment packets. Subsection 2.3 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.

Depending on 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 2 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,

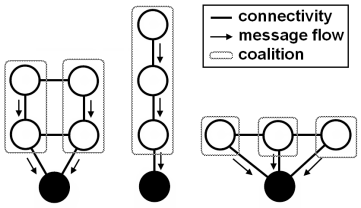


Fig. 2. Examples of routing and coalition formation

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 3.

2.2 Reinforcement learning approach: Methodology

Each agent in the WSN uses a reinforcement learning (RL) [15] 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 [16] with an implicit exploration strategy, as subsection 2.5 will further elaborate on. However, our update scheme differs from that of traditional Q-learning (cf. subsection 2.4). 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 described in this section.

2.3 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 (or single-state). 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 [6]. 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 3.1 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 2.5.

2.4 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 [16], where only the Q-value of the selected action is updated. The motivation behind this update scheme is presented in subsection 2.5. 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 behavior makes neighboring nodes (de)synchronize their actions, resulting in faster message delivery and thus lower end-to-end latency.

2.5 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 [15].

3 RESULTS

We proceed with the experimental comparison between our (de)synchronization approach and a fully synchronized state-of-the-art MAC protocol, viz. S-MAC [17]. All components of the compared networks, such as the routing and CSMA communication protocols, remain the same. The S-MAC protocol illustrates network performance under synchronized behavior, where all nodes are active at the same time. 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, but as we will demonstrate in subsection 3.2, it fails at short duty cycles.

3.1 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 4-node line (Figure 3(a)) and a 6-node single-hop mesh topology (Figure 4(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 5(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 3.2 will confirm these claims and will elaborate on the obtained results.

Each of the three networks was run for 3600 seconds in the OMNeT++ simulator [11] and results were averaged over 30 runs. This network runtime was sufficiently long to eliminate any initial transient effects. To illustrate the performance of the network at high data rates, we set the sampling period of nodes to one message every 10 seconds. For each node the start of this period is at a uniformly random time within the first frame of the simulation and thereafter messages in that node are periodically generated at the same slot every frame. Frames have the same length as the sampling period and were divided in $S = 2000$ slots of 5 milliseconds each. The duration of the slot was chosen such that only one DATA packet can be sent and acknowledged within that time. All hardware-specific parameters, such as transmission power, bit rate, etc., were set according to the data sheet of our radio chip — CC2420 [1]. In addition, we chose the protocol-specific parameters, such as packet header length and number of retransmission retries as specified in the IEEE 802.15.4 communication protocol [3].

Since collisions constitute the biggest obstacle in the pursuit of low latency, each node contends for the channel for a small random duration within a fixed contention window of 5 slots. To facilitate the throughput of messages at high data rates, we deviated from the contention policy of S-MAC that uses the entire active time as a contention window. Instead, in our simulations we fixed the maximum contention window of S-MAC to 5 slots for a more fair comparison.

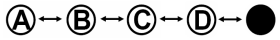
We modeled five different events, namely overhearing ($r = 0$), idle listening ($r = 0$ for each idle slot), successful transmission ($r = 1$ if ACK received), unsuccessful transmission ($r = 0$ if no ACK received) and successful reception ($r = 1$). Maximizing the throughput requires both proper transmission 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 [5], 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).

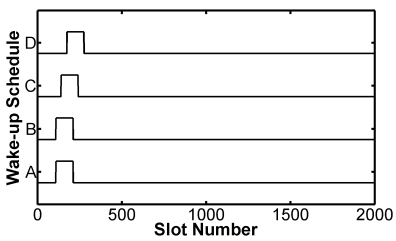
Due to the exponential smoothing nature of the reward update function (cf. subsection 2.4) the Q-values of slots will be shifted towards the latest reward they receive. We would expect that the "goodness" of slots will decrease for negative events (e.g. transmission was not acknowledged), and will increase 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.

3.2 Evaluation

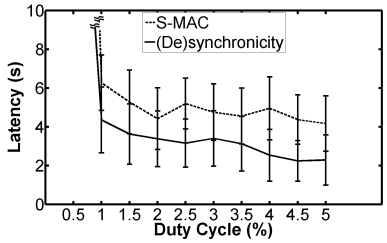
We would like to point out that both S-MAC and our approach are controlled by the same parameter — the duty cycle, which is fixed by the user of the system. Since the active time of nodes in both approaches is the same, the energy consumption of the two protocols is nearly identical. The only difference to S-MAC is that with our approach nodes *learn when* to hold their duty cycle within the frame, as opposed to S-MAC, where all nodes are awake at the beginning of the frame. Therefore, in the following evaluation we vary the duty cycle of the nodes and monitor the average end-to-end latency across the different simulation runs.



(a) Line topology.



(b) Example of a learned wake-up schedule for duty cycle of 5%.



(c) Average end-to-end latency for different duty cycles.

Fig. 3. Experimental results for the line topology

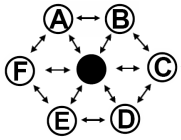
Figure 3(b) displays an example of the resulting schedule of the line topology (Figure 3(a)) after the action of each agent converges for 5% duty cycle. The results indicate that all four nodes have successfully learned to stay awake at the same time in order for messages to be properly forwarded to the sink. In other words, we observe that all nodes belong to the same coalition, as suggested in

Figure 2. If any one node in the line topology had remained active during the sleep period of its immediate neighbors, 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 the overhearing of higher hop communication and to increase throughput by compensating for propagation delays — a behavior that nodes have learned by themselves.

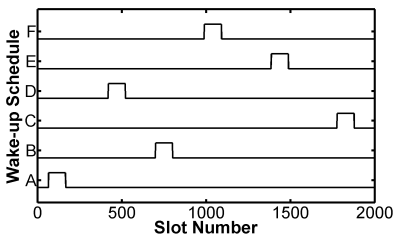
Figure 3(c) displays the average end-to-end latency of the learning and the synchronized nodes respectively, where error bars signify one standard deviation across 30 runs. Since the learned wake-up schedules of our approach closely resemble the prescribed behavior of S-MAC, the latency improvement over different duty cycles is marginal. Nevertheless, the end-to-end latency of our learning agents is on average 2 seconds less than under the S-MAC protocol. The reason for this improvement lies in the fact that with S-MAC all nodes wake up at the beginning of the frame, while with our approach agents learn when it is best to wake up. Since each node periodically generates messages at a different time within the 10-seconds frame, the latency of S-MAC is on average 5 seconds. Learning, however, allows flexibility in the wake-up times, such that a node could wake up immediately after generating its messages (and all other nodes will learn to wake nearly at the same time) and therefore reduce the queuing time of messages for at least one node.

As evident in Figure 3(c), both approaches are inefficient at very low duty cycles. The reason for this high latency is the fact that the active period of nodes is too short compared to the propagation delay. Therefore, messages need to be queued for more than one frame on average, which results in traffic congestion.

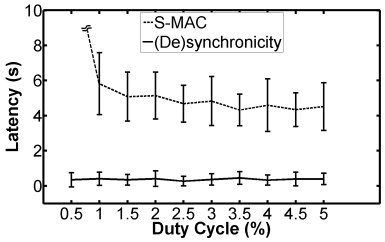
In contrast to the previous topology, our second set of experiments investigate the performance of the network where all nodes lie on the same hop from the sink. This setup presents agents with the opposite challenge, namely to find an active period where no other node is awake. The latter behavior will eliminate communication interference with neighboring nodes and will ensure proper reception of messages at the sink. Figure 4(b) displays an example of the wake-up schedule of the learning nodes for a duty cycle of 5% 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 4(c) where we compare it to the average end-to-end latency of the synchronized system. Error bars represent one standard deviation across 30 runs. Since all nodes lie within one hop of the sink, the performance of the learning agents is not dependent on the duty cycle for this topology. Each node independently learns to hold its active period immediately after it generates a message, as long as no neighbor is awake at the same time. Therefore, the average end-to-end latency is slightly more than the duration of one transmission. Similarly to the line topology, when nodes use the S-MAC protocol, the end-



(a) Mesh topology.



(b) An example of a learned wake-up schedule for duty cycle of 5%.



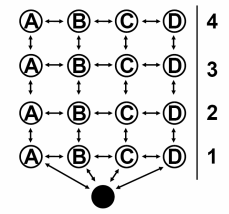
(c) Average end-to-end latency for different duty cycles.

Fig. 4. Experimental results for the mesh topology

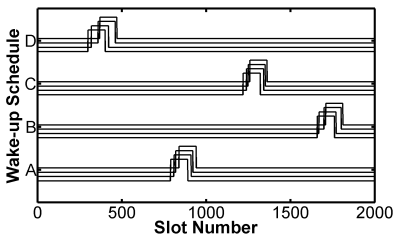
to-end latency of the system is on average half the sampling period, for reasons outlined above. Moreover, for duty cycle of 0.5%, the S-MAC nodes are unable to deliver their messages to the sink, since all nodes try to transmit during the same short awake period and thus all messages collide. This effect is indicated with the discontinued dashed line in Figure 4(c).

Lastly, we investigate a combination of the above two topologies, namely the grid shown in Figure 5(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 routing branches to avoid communication interference. An example of the wake-up schedule of the learning nodes at 5% duty cycle is displayed in Figure 5(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 the same hop have learned to desynchronize their active times similar to the mesh topology.

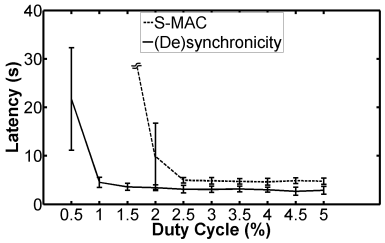
The result of applying our learning approach in a grid topology for various duty cycles can be observed in Figure 5(c). It displays the average end-to-end latency of the network when using synchronicity and (de)synchronicity respectively. Here again error bars signify one standard deviation across 30 runs. Due to the high data rate, when using S-MAC nodes are incapable of delivering all packets for duty cycles lower than 2%. This reduced performance at low duty cycles is due to the large number of collisions and re-transmissions necessary



(a) Grid topology.



(b) An example of a learned wake-up schedule for duty cycle of 5%.



(c) Average end-to-end latency for different duty cycles.

Fig. 5. Experimental results for the grid topology

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. When nodes coordinate their actions, they effectively reduce communication interference with neighboring nodes. This behavior results in lower amount of overheard packets, less collisions and therefore fewer retries to forward a message, as compared to the fully synchronized network. Nevertheless, at very low duty cycles the active time of nodes is too short to forward all messages and therefore, similar to the line topology, the network experiences traffic congestion.

3.3 Discussion

We would like to discuss here the convergence rate of the learning agents. The implicit exploration scheme, described in subsection 2.5 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 80 iterations (or frames) on average the actions of agents do not change and thus the state of (de)synchronicity has been reached. In other words, after 800 seconds each node finds the wake-up schedule that improves message throughput and minimizes communication interference. This duration is sufficiently small compared to the

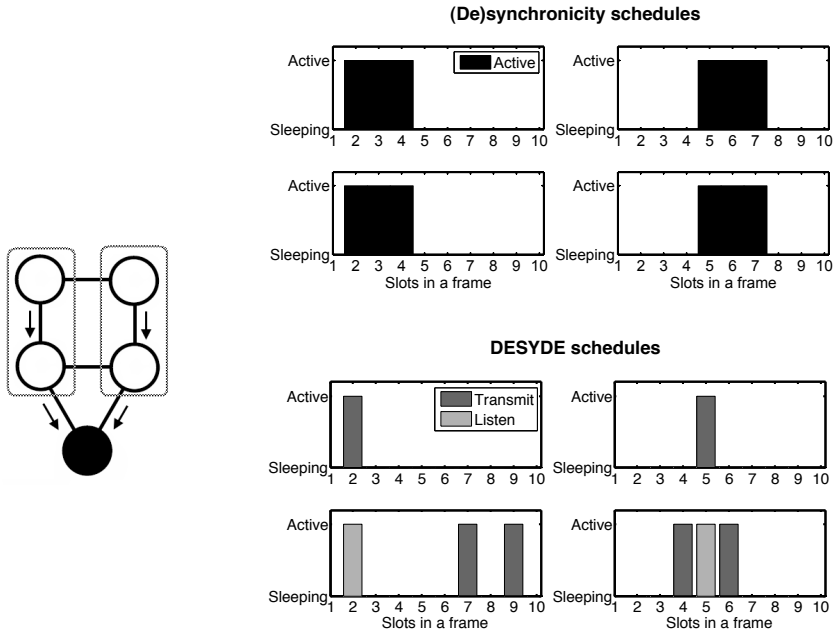
lifetime of the system for a static WSN, which is in the order of several days up to a couple of years depending on the duty cycle and the hardware characteristics [4]. However, it is still unclear under which conditions convergence proofs can be brought. Further research is therefore required to better characterize the convergence criteria.

Despite the improvements that our approach offers over the standard S-MAC protocol, we discuss here two major shortcomings that need to be addressed. First of all, the duty cycle set by the user of the system affects all nodes equally. In other words, all nodes are active for the same amount of time. Depending on their position in the network, however, nodes require different duration for their active periods. Nodes close to the sink are subject to heavier traffic load compared to leaf nodes, whose active time need not be as high. The second shortcoming of our technique concerns the coordination of actions among active agents. Clearly, being awake at the same time is not sufficient for two nodes to successfully exchange messages. If two agents on the same routing branch attempt to transmit at the same slot, their messages will collide. Agents therefore need to learn not only the time of their active period within a frame, but also when to transmit and when to listen during that active period.

The above two shortcomings are being addressed in an extension of our algorithm, which we call DESYDE [9]. The three main differences to the proposed approach are outlined below:

1. In DESYDE we let agents learn two quality values for each slot, instead of one. One quality value indicates how beneficial it is for the node to *transmit* during that slot, while the other value indicates how good it is to *listen* for messages. In slots where it is neither good to transmit nor to listen, the node will turn off its antenna and enter *sleep* mode. Thus, each node learns the quality of three actions: *transmit*, *listen* and *sleep*, as opposed to only *wake-up* and *sleep*.
2. The algorithm in DESYDE differs from the one proposed in this paper also in the value of the learning rate α. In DESYDE we set this value to 1, which dramatically alters the learning behavior of nodes. With $\alpha = 1$, nodes remember only the most recently observed feedback signal for each slot and discard old observations. In this way the behavior of nodes resembles a Win-Stay Lose-Shift strategy [13] where in our setting agents at each slot repeat the action that was successful at the same slot in the previous frame and try a different action if it was unsuccessful.
3. The last difference is the action selection method — in DESYDE nodes select at each slot the action with the highest expected reward, rather than staying awake for the slots with the highest sum of Q-values. If none of the two quality values are above 0 for a given slot, the agent selects *sleep* in that slot in the next frame. In this way nodes adapt their duty cycle to the traffic load of the network and may wake up at different slots within a frame, as opposed to holding only one active period.

To illustrate the effect of the above three differences, consider the 2 by 2 grid in Figure 6(a). An example of the resulting wake-up schedule of the four



(a) 2-by-2 grid topology. (b) Example of (De)synchronicity schedules (top) and DESYDE schedules (bottom) for the 2-by-2 grid.

Fig. 6. Comparison between (De)synchronicity and DESYDE schedules on the 2-by-2 grid.

nodes is illustrated in Figure 6(b) for both (De)synchronicity (top) and DESYDE (bottom). In this example, the frame contains 10 slots, and the four schedules reported (for each approach) are those of the four nodes in the grid, arranged in the same order as in Figure 6(a). Figure 6(b) (top) shows the schedules of nodes using the algorithm presented in this paper. One can see that the left nodes are synchronized for communication at slots $2-4$, while the right ones are active at slots $5-7$. In other words, upper nodes are synchronized with lower ones (being active at the same slots) and left nodes are desynchronized with right.

The schedules of nodes when using DESYDE are presented in Figure 6(b) (bottom). Here upper transmission slots are synchronized with lower reception slots and left active slots are desynchronized with right active slots. More precisely, at slot 2, the upper left node transmits when the lower left node receives, while the right nodes are synchronized for communication at slot 5. The lower left node sends its data to the base station at slot 7 and forwards that of the upper left node at slot 9. The lower right node does the same at slots 4 and 6, respectively. Thus, with both approaches we observe the same coalitions as in our schematic model in Figure 6(a). However, DESYDE allows nodes to adapt their active time based on the traffic load and therefore prolongs the network lifetime.

Moreover, when using DESYDE nodes are able to coordinate their communication more precisely and avoid collisions, resulting in lower end-to-end latency. For a more detailed description of the latter results we refer the reader to [9].

Different axes may finally be considered to improve the efficiency of the proposed (de)synchronization policy. Our protocol is not well suited for irregular data traffic, as such type of traffic is likely to impair the convergence rate of the RL approach. 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.

4 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 improves the throughput of the system even for very low duty cycles, as compared to the standard S-MAC protocol. When using our RL policy, agents independently learn to synchronize their active periods with nodes on the same routing branch, so that message throughput is improved. At the same time, nodes desynchronize with other routing branches in order to reduce communication interference. We demonstrated how initially randomized wake-up schedules successfully converge to the state of (de)synchronicity based only on local interactions and without any form of explicit coordination. As a result, our approach makes it possible that sensor node coordination *emerges* rather than is *agreed* upon.

The proposed approach provides a basis for a number of extensions that we are currently investigating. In particular, the wake-up schedules of individual nodes may be adapted on the basis of their own traffic load, as illustrated by the DESYDE strategy. This adapted version of the protocol allows to further reduce the convergence time and the end-to-end latency of the system.

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 DiCoMAS (IWT60837); and by the Research Foundation - Flanders, Belgium (FWO), project G.0219.09N.

References

1. CC2420: Data sheet: http://focus.ti.com/docs/prod/folders/print/cc2420.html
2. Cohen, R., Kapchits, B.: 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 (2009)

3. Gutierrez, J., Naeve, M., Callaway, E., Bourgeois, M., Mitter, V., Heile, B.: IEEE 802.15. 4: a developing standard for low-power low-cost wireless personal area networks. Network, IEEE 15(5), 12–19 (2002)
4. Ilyas, M., Mahgoub, I.: Handbook of sensor networks: compact wireless and wired sensing systems. CRC (2005)
5. Langendoen, K.: Medium access control in wireless sensor networks. Medium access control in wireless networks 2, 535–560 (2008)
6. Leng, J.: Reinforcement learning and convergence analysis with applications to agent-based systems. Ph.D. thesis, University of South Australia (2008)
7. Liang, S., Tang, Y., Zhu, Q.: Passive wake-up scheme for wireless sensor networks. In: Proceedings of the 2nd ICICIC. p. 507. IEEE Computer Society, Washington, DC, USA (2007)
8. Liu, Z., Elhanany, I.: Rl-mac: a reinforcement learning based mac protocol for wireless sensor networks. Int. J. Sen. Netw. 1(3/4), 117–124 (2006)
9. Mihaylov, M., Le Borgne, Y.A., Tuyls, K., Nowé, A.: Distributed cooperation in wireless sensor networks. In: Yolum, Tumer, K., Stone, P., Sonenberg (eds.) Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011). To appear. Taipei, Taiwan (May 2011)
10. Mihaylov, M., Le Borgne, Y.A., Tuyls, K., Nowé, A.: Self-organizing synchronicity and desynchronicity using reinforcement learning. In: Filipe, J., Fred, A. (eds.) Proceedings of the 3rd International Conference on Agents and Artificial Intelligence (ICAART). pp. 94–103. Rome, Italy (January 2011)
11. OMNET++: Project website: http://www.omnetpp.org/ – a C++ simulation library and framework
12. Paruchuri, V., Basavaraju, S., Durresi, A., Kannan, R., Iyengar, S.S.: Random asynchronous wakeup protocol for sensor networks. In: Proceedings of the 1st BROADNETS. pp. 710–717. IEEE Computer Society, Washington, DC, USA (2004)
13. Posch, M.: Win-Stay, Lose-Shift Strategies for Repeated Games–Memory Length, Aspiration Levels and Noise. Journal of theoretical biology 198(2), 183–195 (1999)
14. Schurgers, C.: Wireless Sensor Networks and Applications, chap. Wakeup Strategies in Wireless Sensor Networks, p. 26. Springer (2007)
15. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
16. Watkins, C.: Learning from delayed rewards. Ph.D. thesis, University of Cambridge, England (1989)
17. Ye, W., Heidemann, J., Estrin, D.: Medium access control with coordinated adaptive sleeping for wireless sensor networks. IEEE/ACM Trans. Netw. 12(3), 493–506 (2004)