Distributed Cooperation in Wireless Sensor Networks

Mihail Mihaylov
Vrije Universiteit Brussel
Pleinlaan 2
Brussels, Belgium
mmihaylo@vub.ac.be

Yann-Aël Le Borgne
Vrije Universiteit Brussel
Pleinlaan 2
Brussels, Belgium
yleborgn@vub.ac.be

Karl Tuyls
Maastricht University
Minderbroedersberg 6a
Maastricht, The Netherlands
k.tuyls@maastrichtuniversity.nl

Ann Nowé
Vrije Universiteit Brussel
Pleinlaan 2
Brussels, Belgium
ann.nowe@vub.ac.be

ABSTRACT

We present a game-theoretic self-organizing approach for scheduling the radio activity of wireless sensor nodes. Our approach makes each node play a *win-stay lose-shift* (WSLS) strategy to choose when to schedule radio transmission, reception and sleeping periods. The proposed strategy relies only on local interactions with neighboring nodes, and is thus fully decentralized. This behavior results in shorter communication schedules, allowing to not only reduce energy consumption by reducing the wake-up cycles of sensor nodes, but also to decrease the data retrieval latency. We implement this WSLS approach in the OMNeT++ sensor network simulator where nodes are organized in three topologies — line, grid and random. We compare the performance of our approach to two state-of-the-art scheduling protocols, namely S-MAC and D-MAC, and show that the WSLS strategy brings significant gains in terms of energy savings, while at the same time reduces communication delays. In addition, we show that our approach performs particularly well in large, random topologies.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*intelligent agents, multiagent systems, coherence and coordination*; C.2.1 [**Computer Communication Networks**]: Network Architecture and Design—*distributed networks, wireless communication*

General Terms

Algorithms, Performance

Keywords

multiagent learning, collective intelligence, teamwork, coalition formation, coordination, implicit cooperation, emergent behavior

Cite as: Distributed Cooperation in Wireless Sensor Networks, M. Mihaylov, Y-A. Le Borgne, K. Tuyls and A. Nowé, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 249-256.

1. INTRODUCTION

Wireless Sensor Networks (WSNs) are a recent class of networks able to monitor our daily environment with a high spatiotemporal accuracy [10, 2]. WSNs are composed of small sensing devices, also known as wireless sensor nodes, endowed with sensing, processing and wireless communication capabilities. Given the current technological trend, WSNs are envisioned to be mass produced at low cost in the next decade, for applications in a wide variety of domains. These include, to name a few, ecology, industry, transportation, or defense [2].

A typical WSN scenario consists of a set of sensor nodes, scattered in an environment, which report their data periodically to a centralized entity called *base station.* The resources of the untethered sensor nodes are often strongly constrained, particularly in terms of energy and communication. The base station usually possesses much larger resources, comparable to those of a standard laptop or desktop computer [10, 2].

The limited resources of the sensor nodes make the design of a WSN application challenging. Application requirements, in terms of latency, data throughput, or lifetime, often conflict with the network capacity and energy resources. The standard approach for addressing these tradeoffs is to rely on *wake-up scheduling* [10], which consists in alternating the active and sleep states of sensor nodes. In the active state, all the components of a node (CPU, sensors, radio) are active, allowing the node to collect, process and communicate information. In the sleep state, all these components are switched off, allowing the node to run with an almost negligible amount of energy. However, nodes in sleep mode cannot communicate with others, since their radio transmitter is switched off. The fraction of time in which the node is in the active mode is referred to as *duty cycle* [19].

Wake-up scheduling offers an efficient way to significantly improve the lifetime of a WSN application, and is well illustrated by S-MAC, a standard synchronized medium access control (MAC) protocol for WSN [20]. 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 node 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 [19].

In this paper we demonstrate how the performance of a WSN network can be further improved, if nodes not only synchronize, but also *desynchronize* with one another. Desynchronization refers to the term where nodes on different branches of the routing tree are active at different times to avoid radio interference. In WSNs nodes can be logically organized in groups. More precisely, the activity schedules of nodes that need to communicate with one another are synchronized to improve message throughput. We say that those nodes belong to one *coalition*. At the same time, the schedules of groups of nodes which do not need to communicate are desynchronized in order to avoid radio interferences and packet losses. We refer to this type of coordination for short as *(de)synchronization*.

We show that coordinating the activities of the sensor nodes can successfully be done using a win-stay lose-shift (WSLS) strategy, drawn from game theory. We call the approach DESYDE, which stands for DEcentralized SYnchronization and DEsynchronization. The coordination is achieved by rewarding successful interactions (e.g., transmission of a message) and penalizing the ones with a negative outcome (e.g., message loss or overhearing). This behavior drives the sensor nodes to repeat actions that result in positive feedback more often and to decrease the probability of unsuccessful interactions. Nodes that tend to select the same successful action naturally form a coalition. The main benefit of the proposed approach is that global (de)synchronization emerges from simple and local interactions without the need of central mediator or any form of explicit coordination. An additional advantage is that DESYDE works with any routing algorithm that forms a routing tree connecting the nodes to the base station.

We implement DESYDE in the OMNeT++ simulator [9], and study three different wireless sensor network topologies, namely line, grid, and random. We compare it to S-MAC [20] and D-MAC [12], two state-of-the-art coordination mechanisms for WSNs, and show that nodes form coalitions which improve data communication and reduce packet collisions. This enables a quicker delivery of the data packets to the base station, allowing shorter active periods and lower energy consumption.

The rest of the paper is organized as follows: Section 2 presents the background of our research. It outlines the application domain, explains the communication and routing protocols and guides the reader through related work. The DESYDE approach is described in Section 3, and experimentally compared on different topologies in Section 4. We finally discuss the results in Section 5 shortly before we conclude in Section 6.

2. BACKGROUND AND RELATED WORK

A Wireless Sensor Network is a collection of densely deployed autonomous devices, called *sensor nodes*, which gather data with the help of sensors [10, 2]. The untethered nodes use radio communication to transmit sensor measurements to a terminal node, called the base station or *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.

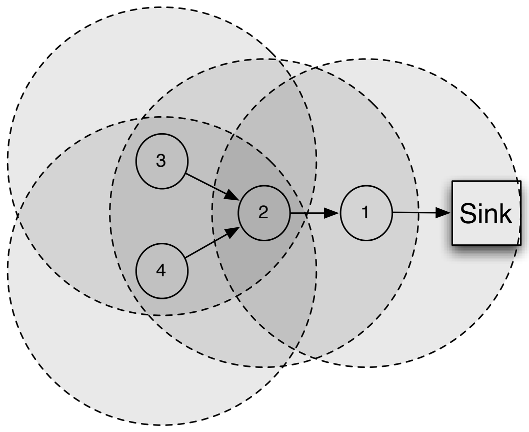


Figure 1: Sensor nodes connected to a base station by means of a multi-hop routing tree. Grayed circles indicate overlapping communication regions.

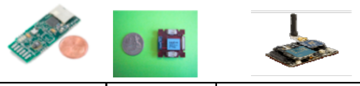
When the WSN is deployed, the routing protocol requires that the nodes determine a routing path to the sink [3, 10]. This is achieved by letting nodes broadcast packets immediately after deployment in order to discover their neighbors. Nodes in communication range of the sink propagate this information to the the rest of the network. During the propagation process, each node chooses a *parent*, i.e. a node to which the data will be forwarded in order to reach the sink. The choice of a parent can be done using different metrics, the standard one being the hop distance, i.e. the minimum number of nodes that will have to forward their packets [4, 18]. An example of multi-hop shortest path routing structure is given in Fig. 1, together with the radio communication ranges of sensor nodes.

Since wireless sensor nodes operate in most cases on finite energy resource, low-power operation is one of the crucial design requirements in sensor networks [2, 10]. The challenge of energy-efficient operation must be tackled on all levels of the network stack, from hardware devices to protocols and applications. Although sensing and data processing may incur significant energy consumption, it is commonly admitted that most of the energy consumption is caused by the radio communication. A large amount of research has therefore been devoted in recent years to the design of energy-efficient communication protocols [10, 20].

Fig. 2 reports the radio characteristics of two representative and often used radio platforms: the CC2420 (used in TelosB and IMote2) and the Xbee-802-15.4 (used in the Waspmote). An important observation is that for these typical radios, the sleep power is at least two orders of magnitude lower than the transmit and receive power. Therefore, the only way to significantly reduce power consumption is to have the radio switched off most of the time, and to turn it on only if messages must be received or sent. This problem is referred to as *wake-up scheduling*.

Wake-up scheduling in wireless sensor networks is an active research domain, and a good survey on wake-up strategies in WSNs is presented in [19]. Three types of wake-up solutions can be identified, namely, on-demand paging, synchronous and asynchronous wake-up.

In on-demand paging, the wake-up functionality is man-



Mote Year	Tmote Sky 2005	Imote 2 2007	Waspmote 2009
Radio	CC2420		Xbee-802.15.4
Outdoor range	50/100m		500 m
Data rate	250 Kbps		20 Kbps
Sleep power	60 μW sleep		<30 μW sleep
Receiving power	63 mW receive		150 mW receive
Transmit power	57 mW wmit		135 mW xmit
Startup time	1 ms setup		2 ms setup

Figure 2: Typical wireless sensor hardware developed in the recent years, together with their main radio characteristics.

aged by a separate radio device, which consumes much less power in the idle state than the main radio. The main radio therefore remains in a sleeping state, until the secondary radio device signals that a message is to be received on the radio channel. This idea was first proposed with the PicoRadio and PicoNode projects [7] for extremely low power systems, and extended in [16, 1] with hand-held devices. On-demand paging is the most flexible and energy-efficient solution, but adds non-negligible costs in the hardware design.

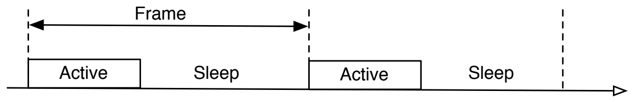


Figure 3: Structure of S-MAC with duty cycle and synchronous wake-up scheduling.

In synchronous wake-up approaches, nodes duty-cycle their radio in a coordinated fashion. Several MAC (Medium Access Control) protocols have been proposed, allowing nodes to wake-up at predetermined periods in time at which communication between nodes becomes possible. A standard paper detailing this idea is that of S-MAC (Sensor-MAC) [20]. The basic scheme is that nodes rely on a fixed duty-cycle, specified by the user, where nodes periodically switch between the active and sleep states. The period is called a frame, and an example of periodic schedule is illustrated in Fig. 3. Several extensions to S-MAC have been proposed. In particular, authors in [12] proposed D-MAC, which aims at improving the efficiency by both reducing the latency and the active period of the sensor nodes. This is achieved by dividing the frame into slots, and by staggering the wake-up cycles along the routing tree, as illustrated in Fig. 4. The active period consists in receiving and sending only one data packet, and active periods are staggered so that nodes send data when their parent's radio is in the receive mode. If multiple packets need to be sent by a node, a *more data* flag is set to warn the parent node that additional data must still be received. If the flag is set, the node and its parent schedule the pending communication to happen after a small backoff period (a few milliseconds), to avoid collisions. This approach was recognized in the MAC review of Langendoen [11] to be particularly compelling for collecting data from a WSN in a timely and energy-efficient manner. The main concern with protocols based on synchronous wake-up is however the overhead which can be caused by maintaining the nodes synchronized.

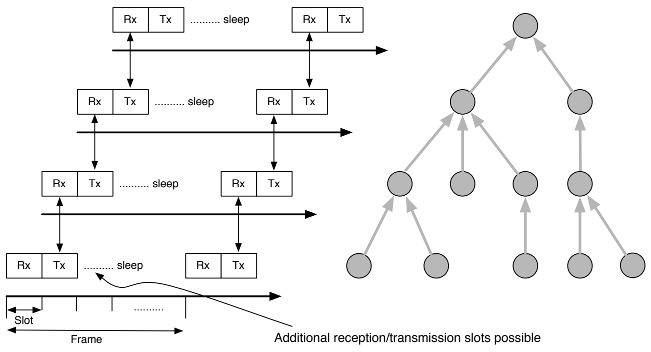


Figure 4: D-MAC scheduling protocol: Nodes are synchronized so that data flows from leaf nodes to the root node in the routing tree.

Finally, in asynchronous wake-up solutions, the wake-up schedules need not be coordinated, and may possibly be different. The communication therefore comes at an increase cost for either the sender or the receiver. In sender-based asynchronous wake-up, the sender continuously sends beacons until the receiver is awake. Once the receiver gets the beacon, it sends an acknowledgment to notify the sender that it is ready to receive a packet. This scheme is the basis for the low-power listening [8] and preamble sampling [5] protocols. The receiver-based wake-up solution is the mirror image of sender-based, and was exposed in the Etiquette protocol [6]. Sender-based and receiver-based asynchronous protocols can achieve very low power consumption. Asynchronous wake-up solutions however require an overhead due to the signaling of wake-up events, which makes them inefficient when wake-up events are relatively frequent [19].

3. DESYDE

This section presents DESYDE, an acronym for DEcentralized SYnchronization and DEsynchronization, which aims at improving communication performances in wireless sensor networks by coordinating the radio activity of neighboring sensor nodes. Our approach belongs to the category of synchronous wake-up strategies, and is intuitively motivated by an important (although not apparent at first sight) weakness of D-MAC. Recall from the previous section that D-MAC schedules the radio activity of sensor nodes in such a way that children and parents in the routing tree synchronize their radio transmission/reception slots. While this strategy appears at first sight to offer great benefits over S-MAC, it only works well if nodes are arranged in a line topology (cf. Fig. 5, middle). Indeed, whenever the routing tree contains several branches, neighboring nodes which are the same number of hops away from the base station may interfere, causing packet losses, and possibly important delays (once a transmission fails in D-MAC, the packet is queued until the next frame).

A better schedule is therefore one where nodes along the same branch of a routing tree are staggered, as in D-MAC (so that end-to-end latency is improved), while at the same time being desynchronized with nodes on neighboring branches of the routing tree (so that communication interference is minimized). Synchronized groups of nodes will be referred

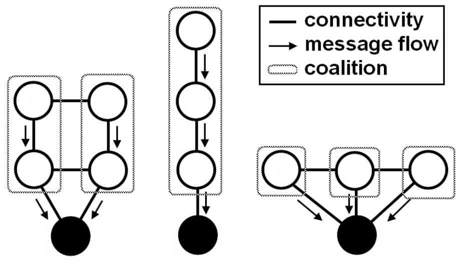


Figure 5: Examples of routing and coalition formation

to as *coalitions*. Figure 5 illustrates the concept of coalitions in three different topologies. Intuitively, nodes on the same branch of a routing tree should form a coalition so that data can be relayed efficiently from a leaf node to the root node. At the same time, nodes which are the same same number of hops away from the base station should desynchronize since they belong to separate branches of the routing tree.

The resulting schedule of DESYDE for the 2 by 2 grid in Fig. 5 (left) is illustrated in Fig. 6. In this example, the frame contains 10 slots, and the four schedules reported are those of the four nodes in the grid, arranged in the same order as in Fig. 5 (left). 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 send 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, we observe the same coalitions as in our schematic model in Figure 5 (left).

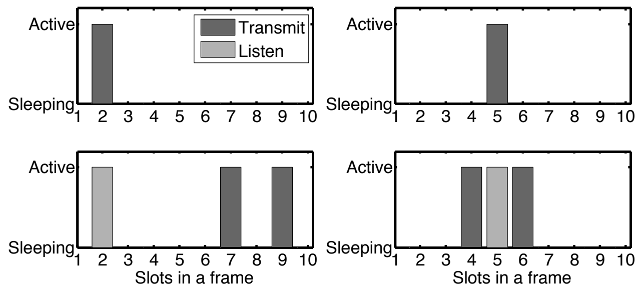


Figure 6: Examples of DESYDE schedule for the 2x2 grid. Upper transmission slots are synchronized with lower reception slots. Left active slots are desynchronized with right active slots.

3.1 Learning Model

We present here the underlying approach that makes nodes coordinate their behavior in a decentralized manner. We see the WSN as a Multi-Agent System (MAS), where agents are the sensor nodes. Recall that the frame stands for the fixed-length activity period that nodes periodically repeat over time. Each frame is divided in a number of slots that constitute discrete time units, where agents can select only one of three actions, namely *transmit*, *listen* or *sleep*.

At each slot, agents are therefore involved in a game with their neighbors where all nodes have to independently choose an action. The goal of agents is to forward all their messages to the sink, minimizing the end-to-end latency of packets. Upon executing any of the three actions, each agent receives feedback from the environment. This feedback depends on the *event* that the actions produced (for example successful transmission, collision, or idle listening). These events will be detailed in section 3.2, and may be either successful or unsuccessful. The success of the resulting event is used by the node to assess the quality of its action.

One can notice that the game our agents are involved in at each time slot has certain characteristics of Graphical Games [17], where neighboring agents have to independently decide on an action. In our network, however, the graphical game at each time step is dependent on the one played at the previous slot, due to forwarding of messages. In other words, our agents are engaged in sequential graphical games, where each game is closely related to the preceding one. The dependence between these sequential games is a result of the forwarding of packets between nodes. Put differently, the transmission of messages between neighbors influences their choice of action at each time step. These actions, however, have no immediate effect on agents further in the network.

Formally, we represent our learning system with the following notation:

- Each frame F is divided in N slots, and the set of time slots in a frame is denoted $S = \{s^1, \dots, s^N\}$.
- A is the set of available actions a for each agent at each slot.
- $R : S \times A \to [0, 1]$ is the reward signal $R(s, a)$ for taking action a in slot s.

As specified above, the action space A at each time slot s for each agent is identical and restricted to the following three actions: $a_{transmit}$, a_{listen} and a_{sleep}. At every time step each agent may detect a communication event, caused by its own actions and those of its neighbors. Upon executing action a, the agent receives a reward $R(s, a)$, determined by the outcome of the actions its neighbors chose at slot s and its own action. This interaction between neighboring nodes is further elaborated in the following subsections.

3.2 Rewards, Updates and Action Selection

We use $Q(s, a)$ to indicate the expected reward (or "quality") of taking action a at slot s. It represents the latest reward obtained at that slot for that action. At first, this value is initialized to 0 for $Q(s, a_{transmit})$ and $Q(s, a_{sleep})$ and 1 for $Q(s, a_{listen})$. Upon executing action a at slot s, the agent updates its action quality, based on the reward it receives: $Q(s, a) \leftarrow R(s, a)$. Note that only one of the three actions can be taken during a slot. Therefore, at every slot s, $Q(s, a)$ is 1 for exactly one action a and 0 for the two others.

We classify each communication event, that a node can detect, using a boolean value to signalize whether the event was positive or negative for that node. Based on our simple update rule a boolean representation is sufficient in our learning model. We modeled six different events, namely successful transmission (if ACK received), successful reception, overhearing, idle listening, unsuccessful transmission (if no ACK received), and collision. We consider these six events to be the most energy expensive or latency crucial in wireless communication. The reward for the two events is 1 (successful transmission or reception) and for the rest it is 0.

Recall that only one action can have a quality of 1 for each slot. We use policy $\pi(s)$ to denote the action a that the agent selects at slot s, based on the quality $Q(s, a)$ for that action in that slot. More precisely,

$$\pi(s) = \begin{cases} a_{transmit}, & \text{if } Q(s, a_{transmit}) = 1 \\ a_{listen}, & \text{if } Q(s, a_{listen}) = 1 \\ a_{sleep}, & \text{if } Q(s, a_{sleep}) = 1 \end{cases} \quad (1)$$

This behavior resembles a *win-stay lose-shift strategy* [15], where agents repeat successful actions and avoid unsuccessful ones. In particular, at slot s an agent will repeat action a only if it had a positive outcome at slot s in the previous frame. Recall that frames capture the periodic behavior of nodes. Thus, in every frame the agent repeats those actions that had positive outcome in the previous frame. For example, according to policy $\pi(s)$, if $Q(s, a_{transmit}) = 1$ for slot s, the node will choose to transmit a packet during that slot (provided that it has a packet in its queue). As a result, a reward $R(s, a_{transmit})$ will be generated and stored in $Q(s, a_{transmit})$. If its transmission was acknowledged, $Q(s, a_{transmit})$ will stay 1 and the agent will repeat the same action next frame at slot s. Otherwise, based on the event that occurred, it will choose a different action. In the same way the agent will select an action in every slot within the frame F.

3.3 Exploration

We would like to note that the outcome of each of the actions $a_{transmit}$ and a_{listen} gives us information about the quality of the other two actions as well. Both these actions require that the radio transmitter of the node is switched on, which enables the agent to detect events in its environment and therefore obtain feedback. Consider the following example. If $\pi(s) = a_{transmit}$, upon observing the outcome the agent will know whether it would have been more beneficial to *listen* or *sleep* during slot s instead. Provided that its message is acknowledged, transmitting is indeed the best action. If no acknowledgment is received, its parent is probably busy (or sleeping) and therefore it would be better to *listen* for packets at slot s next frame, rather than *transmit*. In case it detects a collision during that slot, for next frame *sleeping* would be most beneficial in order to avoid receiving in vain. The same reasoning holds if the agent selects action a_{listen}.

Action a_{sleep} on the other hand turns the energy-consuming antenna off and consequently prevents the agent from receiving any information from its environment. For this reason, during a short exploration stage, fixed by the user, the agent never selects the latter action to keep its radio transmitter on. In other words, during exploration if $\pi(s) = a_{sleep}$ the agent will select a_{listen} at that slot instead. This behavior enables the node to constantly acquire feedback and therefore update its quality values. In the WSN domain, such an exploration is very costly in terms of battery consumption. However, we determined empirically that nodes require no more than 3 to 4 frames of exploration for their policies to converge. After the exploration stage expires, the agent reverts to the policy described in Formula 1. Intuitively, when an agent finds a "win" action for a certain slot, its *win-stay lose-shift* strategy will prevent it from choosing a different action at the same slot next frame. Thus, every agent learns a periodical schedule based on the events that happen as a result of its actions. We therefore say that no explicit form of agent coordination is necessary to achieve equilibrium. Instead, coordination "emerges" as a result of packet forwarding and reasoning based on local interactions.

4. EXPERIMENTAL STUDY

4.1 Experimental Setup

We apply our approach on three networks of different size and topology – a 4-hop line, a 16-node (4 by 4) grid topology and one with 50 nodes scattered randomly with an average of 5 neighbors per node. The first topology 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 forward its messages to the sink. The second topology illustrates the importance of combining synchronization and desynchronization, as neither one of the two behaviors alone is an efficient strategy. The random topology shows the scalability of our approach to larger networks where the topology is not known a priori. In our simulations we use a shortest path routing scheme that creates a static routing tree. A similar experimental setup is also used in [13, 14].

Each of the three networks was ran for 200 seconds in the OMNeT++ simulator [9] 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. To simulate periodic data collection, this message is generated at the beginning of each frame for all nodes. Frames have the same length as the sampling period and were divided in $N = 2000$ 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 (cf. Table 2). 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. We set the duration of the exploration stage to 5 frames, or 50 seconds. It was empirically measured that this duration was enough for the policies of all agents to converge.

To better illustrate the importance and effect of combining synchronization and desynchronization in these topologies, we compare our approach to two state-of-the-art MAC protocols, viz. S-MAC and D-MAC. In addition, we present the case where all nodes remain active for the entire duration of the simulation and never switch off their radio transmitter (called ALL-ON for short). The latter behavior serves only as a benchmark in terms of end-to-end latency, because the energy consumption of this protocol renders it impractical for real-world scenarios. Since nodes have a duty cycle of 100%, packets will not experience any sleep latency and will be quickly forwarded to the sink. The S-MAC protocol illustrates network performance under synchronized behavior, where all nodes are active at the same time. D-MAC on the other hand shows whether staggering the short duty cycles across hops is an efficient strategy to improve latency and lifetime.

4.2 Evaluation

Figures 7 and 8 display the average battery consumption and latency for all three topology using the S-MAC protocol. According to S-MAC, all nodes wake up at the beginning of the frame for a duration specified by the user. We therefore vary the duty cycle and observe the performance of the system in terms of lifetime and throughput. Intuitively, the energy consumption under S-MAC increases linearly with increasing duty cycle for all three topologies, as Figure 7 displays.

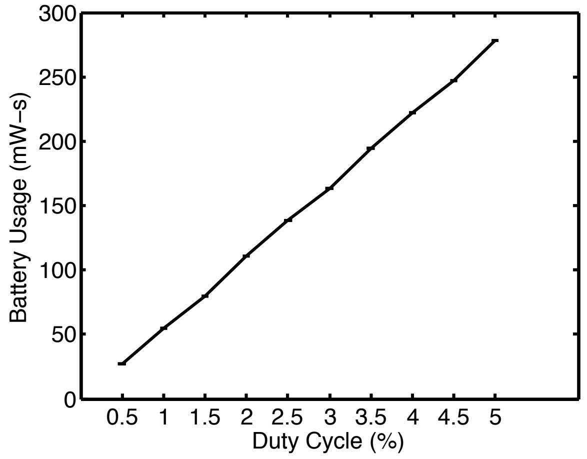


Figure 7: Average battery usage under the S-MAC protocol for different duty cycles (same for all topologies)

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. 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. Even though short duty cycles are appealing from an energy perspective, nodes do not manage to forward all their packets within one active interval for all three topologies. The latter result can be observed in Figure 8, where error bars signify one standard deviation across 30 runs. For duty cycles below 2% (or 1% in the line topology) nodes do not manage to forward all their packets within one active interval. The reason for this high latency is the large number of collisions when all nodes wake up at the same time. This phenomenon is particularly visible in the grid topology, where every node has exactly one parent and at least one neighbor who belongs to a different branch of the routing tree.

For duty cycles larger than 2% nodes in the line and grid topologies manage to send all their packets within one active interval and therefore the average end-to-end latency is reduced to around 0.1 seconds. Nodes in the random network, however, require two active periods to forward their messages and therefore the latency settles at around 10 seconds, or 1 frame duration. Still, we measured around 20% packet loss on average for the latter topology, due to the large number of retransmissions necessary when all nodes are active at the same time. The large standard deviation is due to the different random topologies across runs.

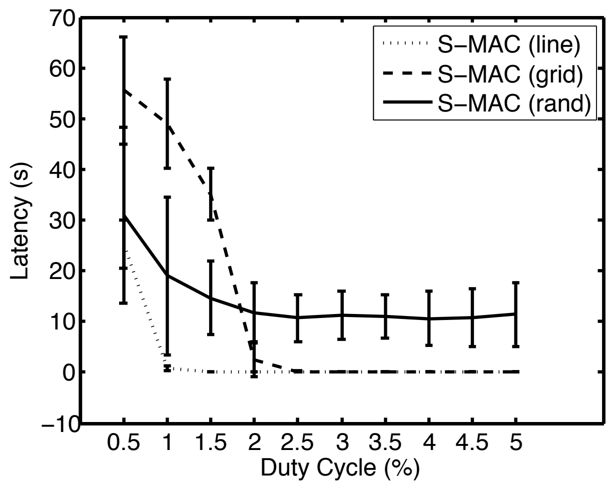


Figure 8: End-to-end latency under the S-MAC protocol for different duty cycles

S-MAC illustrates the effect of full synchronization on the network performance. We see that large networks suffer from high sleep latency for small duty cycles. D-MAC on the other hand makes sure that sleep latency is reduced by "staggering" the wake-up cycles of nodes according to their hop distance to the sink (cf. Figure 4). In other words, all nodes that lie at the same distance from the sink are synchronized to wake up at the same time and send a packet to their parents, who wake up at the slot just after their children. This behavior is designed to reduce the sleep latency that S-MAC suffers from. The duty cycle of each node under D-MAC is dependent on its traffic load (i.e., its position in the data gathering tree), as it is the case with our learning algorithm. Similar to S-MAC, to reduce collisions we let each node contend for the channel for a small random time within a fixed contention window. The size of this window, however, affects performance. Large contention window will lower the probability of collision, but at the cost of delayed transmissions. A small one, on the other hand, will speed up communication, but will cause more unsuccessful transmissions. We therefore present the performance of each protocol for different contention window sizes. To abide by the specifications of D-MAC, we define the size of its contention window in terms of the duration of a DATA packet. The design of DESYDE, however, requires us to set the contention window as a factor of the slot length instead (which is a DATA packet + an acknowledgment). We use the latter setting in S-MAC and ALL-ON as well. Since the difference between the two contention windows is negligible, we use the same axis in Figures 9 and 10 to plot the performance of both protocols. We write "contention window size" to signify the *factor* of the contention window, which is fixed for each run. Due to space limitations we graph the battery consumption of D-MAC and DESYDE for each topology on the same plot.

In Figure 9 we see that in all three topologies DESYDE outperforms D-MAC in terms of energy consumption, irrespective of the contention window size. Due to the "win-stay

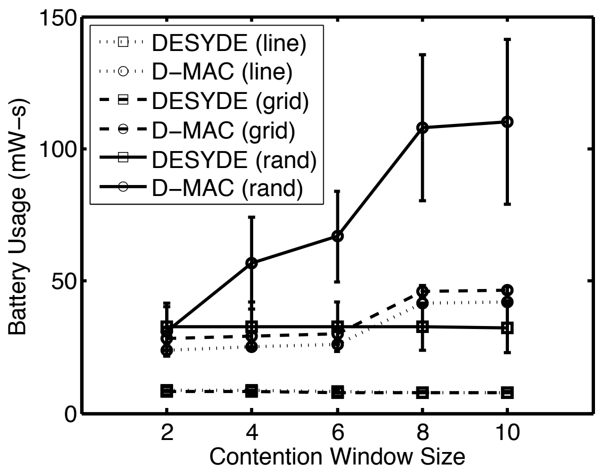


Figure 9: Battery usage under D-MAC and DESYDE for different contention windows across different topologies

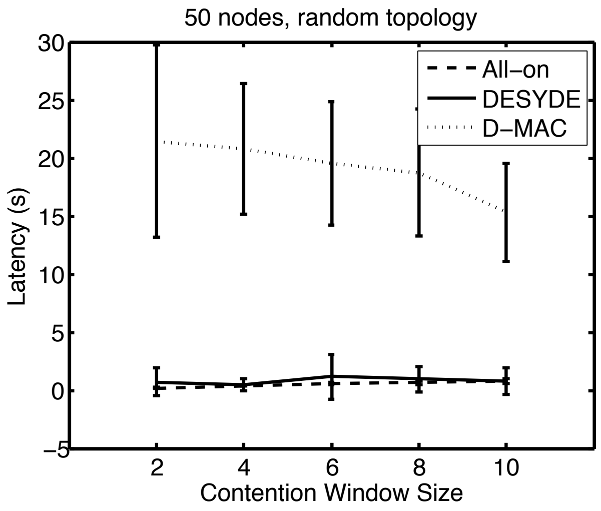


Figure 10: End-to-end latency under ALL-ON, DESYDE and D-MAC for different contention windows

lose-shift" strategy, after the exploration stage our learning algorithm remains invariant to the contention window size. In other words, contention is used only during exploration. Each node, thereafter, learns to transmit in a different time slot within the frame and thus contention for the channel is not necessary. Nodes under D-MAC, however, always wake up for one listen and one transmit slot, regardless of the node position in the network. A disadvantage is that leaf nodes still listen for one slot, when they need not, while all other nodes need to hold an additional listen + transmit slot for every packet they generate. The energy consumption of D-MAC is therefore higher than the one of DESYDE for each topology. Moreover, according to specifications the active period of D-MAC includes the time for channel contention. Therefore its battery consumption increases with the size of the contention window.

Lastly, we present the difference between ALL-ON, DESYDE and D-MAC in terms of the end-to-end latency averaged over 30 random topologies, each consisting of 50 nodes. Figure 10 compares the three protocols for different contention windows. One can notice that DESYDE once again outperforms D-MAC. DESYDE enables nodes to both synchronize with their parents and desynchronize with their same-hop neighbors. Recall the example we presented in Figure 6, which shows the resulting schedule of nodes under DESYDE in a sample 2 by 2 grid topology. In some routing schemes, such as ours, all nodes that lie on the same hop belong to different branches of the routing tree. In D-MAC, however, all those nodes wake up at the same time and therefore cause radio interferences, followed by packet retransmissions. Intuitively, latency under D-MAC decreases for larger contention windows, but nodes still require more than one active period to deliver all their packets. DESYDE, on the other hand, has comparable latency to ALL-ON, where nodes never switch off their antenna and therefore packets incur no sleep delay. While ALL-ON requires 100% duty cycle, DESYDE is able to achieve the same latency with only 0.8% active time within a frame.

5. DISCUSSION AND FUTURE WORK

The experimental results presented in the previous section illustrate that DESYDE is able to significantly improve the performance of a data collection task in wireless sensor networks. The two main metrics considered were the latency and the energy consumption. For both metrics, large gains could be observed, over a wide range of networking parameters. These results were particularly remarkable for large and random topologies. The main reason is that DESYDE relies on a learning strategy which can adapt to complex topologies and traffic patterns.

The *win-stay lose-shift* (WSLS) strategy which underlies DESYDE is a key aspect of the proposed approach. Several research directions can be pursued in order to further improve its performance. First, an advantage of the WSLS is that it provides a way to reduce the exploration space and to accelerate the convergence of the learning stage. A direct drawback of this "aggressive" exploration is that more efficient solutions to the coordination of sensor nodes may be too quickly discarded. One of the research axes we plan to focus on consists in relying on "smoother" updating rules for the quality values of the actions. This could be done by using a learning factor which keeps tracks of past rewards during the learning process.

A second important parameter is the convergence time of the learning process. We observed in all our experiments that this time is in practice very short, in the order of a few data collection rounds (around 5). It is still unclear under which conditions convergence proofs can be brought. One can easily notice that, using the WSLS approach, the convergence is guaranteed if no node modifies its policy for two consecutive rounds. This unfortunately does not seem to be detectable without all nodes exchanging information about their status, which would be energy costly. Further research is therefore required to better characterize the convergence criteria.

Finally, we assumed, as most protocols which fall in the synchronous category, that the traffic patterns and the network topology are stationary for every run. As proposed, DESYDE is not robust to topology changes, or to variations in the data collection rate. The common solutions to these issues is to rely on periodic checks concerning the amount of dropped packets, or queue sizes on the sensor nodes, and to restart the coordination of the nodes if necessary. While not specific to the approach presented in this paper, further research is also required in this area.

6. CONCLUSION

Synchronous wake-up strategies can greatly reduce the duty cycle of sensor nodes in a WSN. We however highlighted in this paper that they suffer from potential high latency and energy waste due to radio interferences and packet collisions. These deficiencies stems from the fact that neighboring sensor nodes should only synchronize their activities when they engage in a communication, and *desynchronize* otherwise. The proposed approach, DESYDE, a DEcentralized SYnchronization DEsynchronization strategy, aims at tackling this problem. The core of the approach is based on a win-stay lose-switch strategy, which we implement in a fully decentralized way.

Our OMNeT++ implementation showed that state-of-the-art synchronized protocols only perform well in simple networks, such as line topologies. As the network complexity grows, these protocols resulted in high latency and energy costs, due to the increased number of packet collisions and packet retransmissions. DESYDE was able in all our experiments to compete with standard approaches, and exhibited significant gains in latency and energy especially for larger networks.

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 (FWO), project G.0219.09N.

7. REFERENCES

[1] Y. Agarwal, R. Gupta, and C. Schurgers. Dynamic power management using on demand paging for networked embedded systems. In *Proceedings of the Asia and South Pacific Design Automation Conference*, volume 2, pages 755–759, 2005.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, 2002.

[3] J. Al-Karaki and A. Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE*, 11(6):6–28, 2004.

[4] D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. *Wireless Networks*, 11(4):419–434, 2005.

[5] A. El-Hoiydi. Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks. In *IEEE International Conference on Communications*, volume 5, pages 3418–3423, 2002.

[6] S. Goel. *Etiquette protocol for ultra low power operation in energy constrained sensor networks*. PhD thesis, New Brunswick, NJ, USA, 2005.

[7] C. Guo, L. Zhong, and J. Rabaey. Low power distributed MAC for ad hoc sensor radio networks. *GLOBECOM*, 5:2944–2948, 2001.

[8] J. Hill and D. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE micro*, 22(6):12–24, 2002.

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

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

[11] K. Langendoen. Medium access control in wireless sensor networks. *Medium access control in wireless networks*, 2:535–560, 2008.

[12] G. Lu, B. Krishnamachari, and C. Raghavendra. An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 224, 2004.

[13] M. Mihaylov, Y.-A. Le Borgne, K. Tuyls, and A. Nowé. Decentralised Reinforcement Learning for Energy-Efficient Scheduling in Wireless Sensor Networks. *International Journal of Communication Networks and Distributed Systems*, 6, 2011. To appear.

[14] M. Mihaylov, Y.-A. Le Borgne, K. Tuyls, and A. Nowé. Self-Organizing Synchronicity and Desynchronicity using Reinforcement Learning. In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence*, pages 94–103, Rome, Italy, 2011.

[15] M. Posch. Win-Stay, Lose-Shift Strategies for Repeated Games–Memory Length, Aspiration Levels and Noise. *Journal of theoretical biology*, 198(2):183–195, 1999.

[16] E. Shih, P. Bahl, and M. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 160–171, 2002.

[17] D. Vickrey and D. Koller. Multi-agent algorithms for solving graphical games. In *Proceedings of the National Conference on Artificial Intelligence*, pages 345–351. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[18] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27. ACM, 2003.

[19] W. W. Y. Li, M.T. Thai, editor. *Wireless Sensor Networks and Applications*, chapter Wakeup Strategies in Wireless Sensor Networks, page 195. Springer, 2008.

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