SLACK-MAC: Adaptive MAC Protocol for Low Duty-Cycle Wireless Sensor Networks

Affoua Thérèse Aby[(1,2)], Alexandre Guitton[(1,2)], Pascal Lafourcade[(3,2)], Michel Misson[(3,2)]
(1) Clermont Université, Université Blaise Pascal, LIMOS, BP 10448, F-63000 Clermont-Ferrand, France
(2) CNRS, UMR 6158, LIMOS, F-63173 Aubière, France
(3) Clermont Université, Université d'Auvergne, LIMOS, BP 10448, F-63000 Clermont-Ferrand, France
Emails: {aby,guitton,lafourcade,misson}@sancy.univ-bpclermont.fr

***Abstract*—Wireless sensor networks (WSNs) are increasingly used in environmental monitoring applications. They are designed to operate for several months by featuring low activity cycles, in order to save energy. In this paper, we propose a MAC protocol for such WSNs with duty-cycles of 1%. Initially, nodes are activated randomly and independently, then they use the knowledge of previous successful frame exchanges to compute their next activation times. We study the choice of the history size, and we compare the performance of our protocol with other protocols from the literature. We show that with a limited history size of only six entries, we significantly improve the performance of existing protocols, while keeping the advantages of fully asynchronous protocols.**

I. Introduction

Environmental monitoring applications, such as the monitoring of volcanoes [**?**], bird nests [**?**], fields [**?**], or bridges [**?**], are increasingly using Wireless sensor networks (WSNs). In such applications, wireless sensor nodes are deployed in the environment, where they perform periodic measurements and communicate the collected data to a sink in a multi-hop manner.

Energy-efficient protocols are designed to increase the lifetime of such WSNs. These protocols deactivate the radio module of nodes most of the time, as it is the node hardware component having the largest energy consumption. The MAC protocol is responsible for allowing nodes to communicate in the rare periods when the radio module of neighbor nodes is active too.

In this paper, we propose the SLACK-MAC (*Self-adaptive Low Activity Cycle Knowledge-based* MAC) protocol. SLACK-MAC is an asynchronous protocol where nodes activate their radio modules randomly. The idea behind SLACK-MAC is inspired by the routing protocol proposed in [**?**], where authors proposed the SR3 protocol, which is an improvement over a biased random walk based on a reputation mechanism. In SLACK-MAC, nodes communicate opportunistically and consider discrete time. Nodes record a history of previous successful communications with neighbors, and use this history to determine the time of the next activation of their radio module. This history increases the probability to select a recent successful time of activation among all possible times during each cycle. Thus, nodes adapt their activation times depending on their neighborhood. We show in this paper that this behavior improves the probability of successful communications, which in turn improves the performance in terms of delivery rate and delay.

The remainder of this paper is organized as follows. Section **??** presents the main existing MAC protocols for low duty-cycles. Section **??** describes the SLACK-MAC protocol, and justifies our choice of parameters. Section **??** compares the performance of existing protocols with the performance of SLACK-MAC. Finally, Section **??** concludes our work.

II. State of the art

Most energy-efficient MAC protocols for WSNs are based on sequences of active and inactive periods, called *duty-cycle*. Indeed, as the radio module of a node is the component having the largest energy consumption, energy can be saved by deactivating it periodically. MAC protocols based on duty-cycles can be classified depending on whether the activities of nodes are synchronized or not.

A. Synchronous MAC protocols

In synchronous duty-cycle MAC protocols, nodes share a common time (through synchronization) and agree on a common schedule for their activities and inactivities. Generally, all nodes are either simultaneously active or simultaneously inactive.

The IEEE 802.15.4 standard [**?**] in beacon-enabled mode is one of the most largely used synchronous MAC protocols. Full-function devices send periodic beacons, with period BI (for Beacon Interval). Reduced-function devices start their activities at the beacon reception, and are allowed to communicate during a period SD (for Superframe Duration). The communication is performed using the slotted CSMA/CA (Carrier-Sense Multiple Access with Collision Avoidance) mechanism, which is designed to consume low energy for channel sensing. After this period, nodes go back to sleep until the next beacon. The ratio SD/BI defines the duty-cycle of nodes.

Several other protocols have been proposed for the same purpose, such as D-MAC [**?**], DW-MAC [**?**], Speed-MAC [**?**], TreeMAC [**?**], MC-LMAC [**?**], SEA-MAC [**?**] and others [**?**], [**?**], [**?**]. As in IEEE 802.15.4, these protocols generally have three types of periods: a *synchronization period*, which ensures that all nodes share a common time, a *communication period*,

where nodes can communicate efficiently, and an *inactive period*, where nodes save energy.

Synchronous duty-cycle MAC protocols have two main drawbacks. The first drawback is the overhead of the mandatory synchronization period. The second drawback is the high contention for the channel when all nodes are active simultaneously. In this paper, we focus on asynchronous duty-cycle MAC protocols.

B. Asynchronous MAC protocols

Asynchronous duty-cycle MAC protocols do not need to synchronize nodes. Notice that generally, asynchronous MAC protocols yield large delays, but have a low energy consumption. In the following, we describe the two main categories of asynchronous protocols: sender-initiated protocols and receiver-initiated protocols.

1) Sender-initiated protocols: The first asynchronous MAC protocol for WSNs was B-MAC [**?**], which is based on the LPL (Low Power Listening) technique. In B-MAC, the source sends a long preamble before each frame, and receivers wake up periodically to detect potential preambles. This technique has been the basis for sender-initiated protocols.

X-MAC [**?**], [**?**] is based on a similar approach. In X-MAC, nodes switch between active (20 ms) and inactive period (500 ms), but instead of using a long preamble, nodes send small preambles to inform the receiver. The maximum duration of the series of short preambles is one inactivity period (500 ms). Once the receiver wakes up and receives a short preamble, it replies by an acknowledgment to inform the transmitter of its availability to receive data, as shown in Figure **??**. When a node having no packet to send wakes up and hears a preamble for another node, it immediately returns to sleep. When a node having packets to send wakes up and hears another preamble, it stops sending its own preamble and waits to receive the acknowledgment for the other transmission before attempting to send its own preamble again. In X-MAC, some source nodes remain active much longer than other nodes. This causes an inequity in energy consumption which reduces network lifetime. Moreover, X-MAC generally achieves a low end-to-end delay, but increases the risk of collisions due to the fact that nodes can interpret the duration between two preambles as a free channel.

In WiseMAC [**?**], preambles are used, but their length is reduced by allowing the sender to send data as soon as both nodes are active.

In [**?**], the authors proposed a distributed algorithm to control the sleep interval of nodes in order to achieve fairness of energy consumption in asynchronous duty-cycle WSNs. The mechanism can increase network lifetime, but has a significant impact on the end-to-end delay.

In this paper, we decide not to use a sender-initiated approach, in order to avoid the overhead and energy consumption caused by preambles. Indeed, for very low duty-cycles, the average duration for a preamble is long.

2) Receiver-initiated protocols: In RI-MAC [**?**], the receiver initiates the communication by sending a beacon to express its ability to receive data packets. RI-MAC reduces channel occupation (as it does not require nodes to send preambles), but introduces a wasted period as the sender has to wait for the reception of the beacon. The ABD protocol [**?**] adds a broadcast service to the RI-MAC protocol.

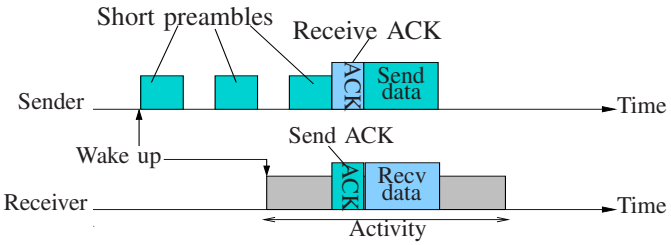

Figure 1. X-MAC's short preamble approach.

In PW-MAC [**?**], each node computes its awakening times according to a pseudo-random number generator rather than according to a fixed schedule. The drawback of PW-MAC is that sending beacons before frame transmissions generates overhead, and introduces a delay when listening to the channel.

In EM-MAC [**?**], nodes decide independently their wake-up time schedule and channel using a pseudo-random generator. EM-MAC allows the sender to wake up just before the beacon of the receiver. However, EM-MAC requires a neighbor discovery phase before starting and each node needs to maintain information about all its neighbors.

HKMAC [**?**] uses an hybrid approach, where time is divided into random activation periods (similar as in RI-MAC) and scheduled activation periods (which requires synchronization).

The MAC protocol proposed in [**?**] is based on random wake-up times. Each node knows the duration of the cycle, denoted by C, and the duration of its activity within each cycle, denoted by A. Each node activates its radio module during A time units every C time units. The beginning of the activation within each cycle is chosen uniformly at random in $[0; C-A[$. When a node is active, it uses unslotted CSMA/CA to access the medium (as in the non beacon-enabled mode of IEEE 802.15.4 [**?**]). With this mechanism, nodes are not synchronized, and nodes do not make assumptions about the activity times of the others. Moreover, there is a non-null probability that any two neighbors share a common activity at each cycle. Figure **??** depicts an example of the activities of three neighbor nodes for this protocol: n_1, n_2 and n_3. We notice that the cycles of nodes are not synchronized. During the first cycle of n_1, nodes n_1 and n_2 share a common activity, during which they can communicate. However, for n_1 to communicate with n_3, both nodes have to wait until the middle of the third cycle of n_1.

In this paper, we focus on a receiver-initiated protocol based on random node activities, as in [**?**].

III. MAC PROTOCOLS FOR LOW DUTY-CYCLES

In this section, we describe our SLACK-MAC protocol and our methodology to choose its parameters.

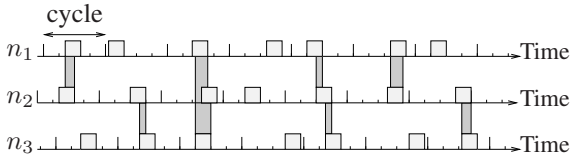

Figure 2. Example of the activities of three neighbor nodes with the protocol of [**?**], with a duty-cycle of 25% (this long duty-cycle is chosen for clarity).

A. SLACK-MAC protocol

The main idea of SLACK-MAC is to maintain a history of times corresponding to successful communications with neighbors. In SLACK-MAC, nodes do not always choose their activation times uniformly at random. Instead, they have a high probability to choose times when successful communications occurred in the recent past. Figure **??** depicts an example of the activities of three neighbor nodes with SLACK-MAC: n_1, n_2 and n_3. Initially, all nodes choose their activation times uniformly at random. When a node chooses a time that yields to successful communications (reception or transmission of a frame), it memorizes it and the probability to choose this time increases.

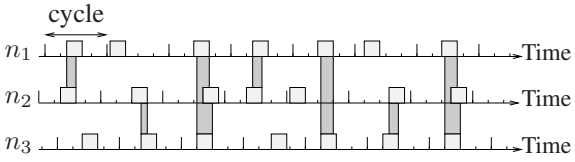

Figure 3. Example of the activities of three neighbor nodes with the SLACK-MAC protocol, with a duty-cycle of 25% (again, this long duty-cycle is chosen for clarity).

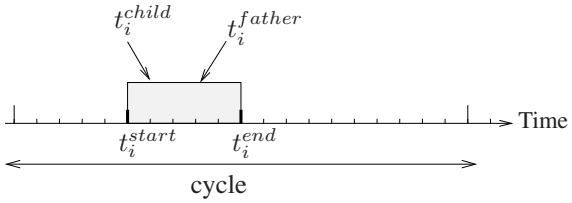

Figure 4. Zoom on a cycle where t_i^{start} and t_i^{end} are respectively activity start time and activity end time in a cycle i with the SLACK-MAC protocol, with a duty-cycle of 25%.

Then SLACK-MAC requires that each node maintains two lists *E* (Emission) and *R* (Reception) that contain wake-up times in the cycle. Let us denote by t_i^{start} the start of activity in the current cycle i, and t_i^{end} is the end of activity time, as depicted on Figure **??**. During this activity a node can send and receive one or several frame, and can add t_i^{start} in both lists, once or several times. Each node uses these two lists to determine its next wake-up time.

A new time t_i^{start} is added to *E* when a node wakes up at time t_i^{start} and communicates with another node located closer to the sink at time t_i^{father}. Similarly, a new time t_i^{start} is added to *R* when a node wakes up in a given cycle and communicates with another node located further away from the sink at time t_i^{child}.

Figure **??** shows the evolution of lists E and R, at three different time steps. Step 1 shows the state of the lists when they are being filled (with one successful transmission and two successful receptions). When a list is full (see Step 2), the last entry is removed to add the newest entry to the front (using a first-in first-out mechanism, as shown on Step 3).

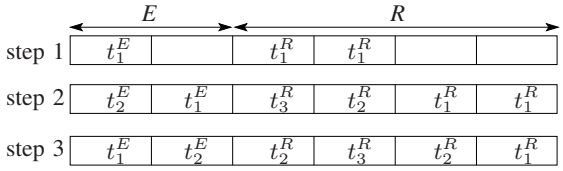

Figure 5. Example of the state of the *E* and *R* lists, at three different times.

More formally, the probability that a node selects its next wake-up time $t \in D$ (where D denotes all possible times) is given by the following formula:

$$Pr[X = t] = \frac{\mathbb{1}_{|R|\neq 0}\left(\frac{|R|_t}{|R|}\right) + \mathbb{1}_{|E|\neq 0}\left(\frac{|E|_t}{|E|}\right) + \frac{1}{|D|}}{\mathbb{1}_{|R|\neq 0} + \mathbb{1}_{|E|\neq 0} + 1},$$

where $|L|$ denotes the number of elements in list L, $|L|_t$ denotes the number of occurrences of time t in L, and $\mathbb{1}_P$ is the indicator function that is 1 when the predicate P is true and 0 otherwise. Note that nodes consider that time is discrete (the granularity of time can be, for instance, 320 μs, as in IEEE 802.15.4, which results into 15,625 slots for a cycle of $C = 5$ seconds).

In Algorithm **??**, we give a pseudo code of the behavior of a node when it wakes up. During this time, a node can receive and send some data. These two operations change the content of *SendQueue* and the two lists *E* and *R*. Before going back to sleep, a node uses these lists to determine its next wake-up time using the function Next-Wake-Up-Time defined in Algorithm **??**, where random(x) draws an integer uniformly at random within $[0, x-1]$, the duration of a cycle is denoted by C time units and the activity of a node by A time units.

The function Next-Wake-Up-Time draws the next wake up time according to the content of the two lists and following the previous distribution. Note that initially, when both lists are empty, we have $Pr[X = t] = \frac{1}{|D|}$, meaning that the next wake-up time is chosen uniformly at random in D. If one of the two lists is empty, we select an element of the non empty list with a probability of $1/2$, and uniformly at random in D otherwise. If none of the list is empty, we select an element from list *R* with probability of 1/3, from list *E* with probability 1/3, and uniformly at random in D otherwise.

Moreover, each node has a packet queue of fixed size for packets that have to be sent, denoted *SendQueue*. If *SendQueue* is full and a node receives a new packet, the node ignores this last packet.

In order to optimize our protocol according to the state of *SendQueue*, each node adapts its selection strategy for its next wake-up time, according to the following rules:

- If *SendQueue* is empty ($state = 1$), a node has no packet to send, so it is useless to select times that are in the *E*

list. The next wake-up time is chosen uniformly in *R* with probability $1/2$, and uniformly at random in D (*i.e.*, all possible times) otherwise.

- If *SendQueue* is full ($state = 2$), a node cannot accept any incoming packet, so it is useless to select times that are in the *R* list. The next wake-up time is chosen uniformly in *E* with probability $1/2$, and uniformly at random in D otherwise.
- In all the other cases ($state = 3$), a node selects its next wake-up time uniformly in *E* with probability of 1/3, uniformly in *R* with probability 1/3, and randomly in D otherwise.

Algorithm 1 Activity of a node.

Node n wake-up at time t_i^{start} for duration A time units in a cycle i of C time units.
while node n (at distance d) is active **do**
  if n has received a frame from n_r (at distance d_r) during cycle i **then**
    if ($d < d_r$) and (t_i^{start} has not yet been added) **then**
      add(n_r, t_i^S) to *R*
      add frame to *SendQueue*
    end if
  end if
  if n has sent a frame to n_s (at distance d_s) during cycle i **then**
    if t_i^{start} has not yet been added **then**
      add(n_s, t_i^{start}) to *E*
      remove frame from *SendQueue*
    end if
  end if
end while
if *SendQueue* is empty **then**
  $t \leftarrow$ Next-Wake-Up-Time(1);
else
  if *SendQueue* is full **then**
    $t \leftarrow$ Next-Wake-Up-Time(2);
  else
    $t \leftarrow$ Next-Wake-Up-Time(3);
  end if
end if
Schedule next activity at time t of the next cycle

Algorithm 2 Next wake-up time in a cycle i.

Next-Wake-Up-Time(state);
if state=1 **then**
  $indicator \leftarrow$ random(2);
  if $indicator = 0$ **then**
    $position \leftarrow$ random(sizeOf(*R*));
    $t \leftarrow$ *R*[$position$];
  else
    $t \leftarrow$ random($C - A$);
  end if
else
  if state=2 **then**
    $indicator \leftarrow$ random(2);
    if $indicator = 0$ **then**
      $position \leftarrow$ random(sizeOf(*E*));
      $t \leftarrow$ *E*[$position$];
    else
      $t \leftarrow$ random($C - A$);
    end if
  else
    $indicator \leftarrow$ random(3);
    if $indicator = 0$ **then**
      $position \leftarrow$ random(sizeOf(*E*));
      $t \leftarrow$ *E*[$position$];
    else
      if $indicator = 1$ **then**
        $position \leftarrow$ random(sizeOf(*R*));
        $t \leftarrow$ *R*[$position$];
      else
        $t \leftarrow$ random($C - A$);
      end if
    end if
  end if
end if
return t

B. Determination of the size of SLACK-MAC lists

In order to determine the size of both *E* and *R* lists, we need to specify the routing algorithm used in our experiments. We have taken a gradient-based routing protocol. Gradient-based routing protocols operate by estimating a distance, called the gradient, to the sink. When a node receives a frame to forward to the sink, the node sends the frame to any neighbor having a gradient smaller than its own gradient. The gradient is computed in the following way: initially, only the sink has a gradient of 0; when a node has a gradient, it sends its gradient to its neighbors; when a node receives a gradient from a neighbor, it updates its own gradient if it detects that this neighbor is closer to the sink than itself. The gradient is generally computed according to several parameters, including the hop count to the sink, link quality estimations, etc.

We first notice that for a routing protocol based on gradient and for a random topology with one sink (located at one corner of the area), a node is likely to have more neighbors further away from the sink than closer to the sink. We estimate that this ratio is about two, which means that the maximum size of *R* is set to be twice the maximum size of *E*. In order to determine the actual size of these lists, we perform 100 simulations over 10 random topologies (of average degree 8) for three different values of the traffic generation period (P).

Figure **??** shows the delivery ratio as a function of the size of *E*, and Figure **??** shows the end-to-end delay as a function of the size of *E*, both with $|R| = 2|E|$. We observe in these two figures that regardless of the period (P), the best size is two for *E* and four for *R* when combining the two criteria. These parameters are used in the following.

We also observed experimentally that on average, it takes

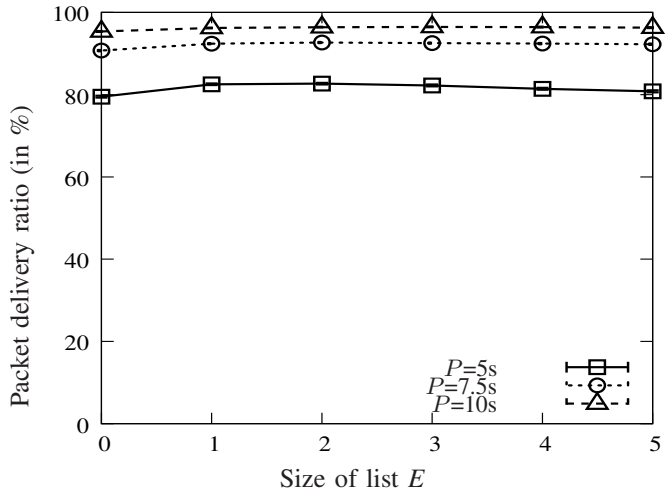

Figure 6. Impact of the size of list E of SLACK-MAC on the packet delivery ratio, with $|R| = 2|E|$ and a duty-cycle of 1%, where P is the traffic generation period.

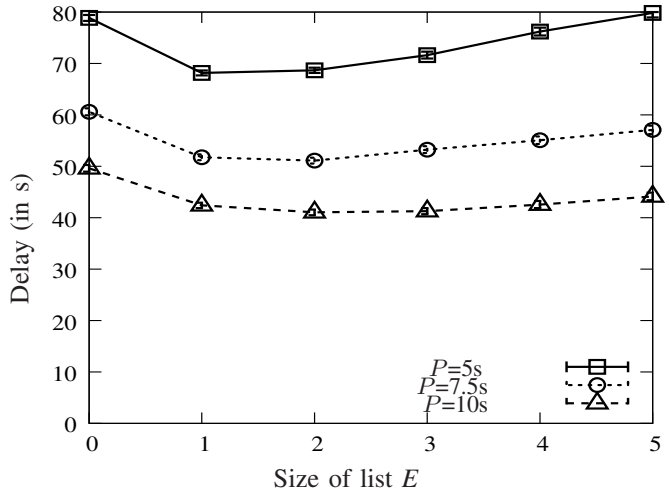

Figure 7. Impact of the size of list E of SLACK-MAC on the end-to-end delay, with $|R| = 2|E|$ and a duty-cycle of 1%, , where P is the traffic generation period.

about 12 cycles (60 seconds) for the nodes to fill E and about 50 cycles (250 seconds) for the nodes to fill R. This shows that the convergence of the list is fast and negligible comparing to the life time of a node.

IV. Results

In order to evaluate the performance of SLACK-MAC, we conducted several simulations to compare SLACK-MAC with the protocol of [**?**] and with X-MAC [**?**] (as it is one of the most representative asynchronous MAC protocols).

We also compared SLACK-MAC with the main MAC standard for synchronized duty-cycle, which is ZigBee [**?**]. ZigBee defines the upper layers of the network stack of a wireless personal area network, and assumes that the lower layers are compliant with IEEE 802.15.4. In the following, we use the tree-based routing protocol of ZigBee (which is used when addresses are allocated hierarchically), as a comparison basis.

A. Simulation parameters

Table I
Simulation parameters

Topologies area	170 m x 170 m
Transmission range	30 m
Number of nodes	100
Number of source nodes	30
Transmission power	0 dBm
Propagation model	shadowing model
Path loss exponent	2.74
Packet size	30 bytes
Maximum send queue size	20
Number of repetitions per topology	100
Simulation duration	3600 seconds

Our simulations are performed using NS-2 [**?**]. The simulation parameters for all protocols are given in Table **??**. In our settings, 30 sources perform periodic measurements and route data (in a multi-hop manner) to a single sink located at one corner of the network. Nodes have a duty-cycle of 1% and the global cycle is 5 s (that is, nodes are active during A=50 ms every C=5 s), unless specified otherwise. For our simulations, we use 10 random topologies of 100 nodes, having a maximum number of hops of 7. All presented results are averaged over 100 repetitions per topology.

It should also be noted that apart from ZigBee [**?**] which incorporates a tree routing protocol, the same gradient-based routing protocol is used to route packets hop by hop towards the sink for all the other MAC protocols.

B. Simulation results

Our objective being to provide a MAC protocol with very low duty-cycle (1%), we initially show that the synchronous duty-cycle MAC protocols are not adapted to such low duty-cycles.

Figure **??** shows the packet delivery ratio as a function of the traffic generation period for ZigBee, X-MAC [**?**], the protocol of [**?**] and SLACK-MAC. The traffic generation period ranges from 5 seconds (which corresponds to a high traffic generation for a duty-cycle of 1%) to 20 seconds (which corresponds to a relatively low traffic generation for such duty-cycle).

For the ZigBee protocol, the packet delivery ratio increases from about 25% to nearly 78%. This low packet delivery ratio with ZigBee is due to the fact that the low duty-cycle generates a strong contention (as nodes are all synchronized). This strong contention generates many collisions and causes overflows in nodes queues, causing a large packet loss ratio. It is also important to note that these results do not take into account the cost of synchronization because we assume that all nodes are synchronized.

For the X-MAC protocol, the packet delivery ratio increases from about 64% to 86%. Although X-MAC does not set a fixed duty-cycle for each node, the delivery ratio is explained by the

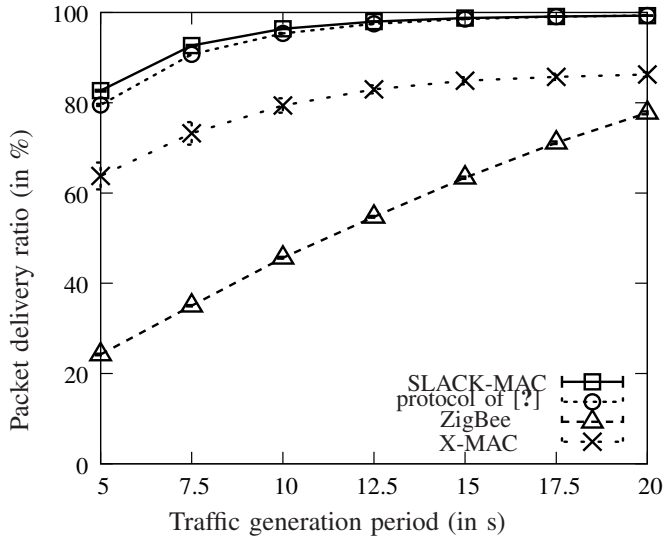

Figure 8. Packet delivery ratio as a function of the traffic generation period, for a duty-cycle of 1%, and for several protocols.

collisions due to the relatively high number of preambles, and by the fact that the sender has no knowledge of the successful reception of packets by the receiver.

For the protocol of [**?**], the packet delivery ratio increases from 79% to 99%. The packet delivery ratio is high: indeed, when nodes meets, they can benefit from this meeting time, as there are few simultaneously active nodes.

For the SLACK-MAC protocol, the packet delivery ratio increases from about 83% to 99%. Indeed, it takes advantage of a mechanism similar to the protocol of [**?**], and allows more common activities between nodes thanks to the history mechanism. SLACK-MAC also remains completely dynamic with a probability of 1/3 for nodes to choose a random mechanism. The results show that, in terms of packet delivery ratio, for traffic generation period from 5 seconds to 20 seconds, SLACK-MAC provides a gain over X-MAC of 29.61% for a period of 5 seconds and of 15.10% for a period of 20 seconds, a gain over ZigBee of 241.26% for a period of 5 seconds and of 27.70% for a period of 20 seconds, and a gain over the protocol of [**?**] of 3.98% for a period of 5 seconds and the same delivery ratio for a period of 20 seconds.

Figure **??** shows the average delay of data packets as a function of the traffic generation period (from 5 seconds to 20 seconds) for ZigBee, X-MAC, the protocol of [**?**] and SLACK-MAC. We note that SLACK-MAC provides a lower end-to-end delay (going from 25 seconds to 68 seconds) than ZigBee (from 37 seconds to 205 seconds) and the protocol of [**?**] (from about 29 seconds to 79 seconds). The delay of X-MAC is very low (from 6 seconds to 11 seconds) compared to the other protocols, at the cost of a high energy consumption (see Figure **??**, described later). Indeed, when a sender has frames to send, it remains active. SLACK-MAC provides a gain over ZigBee in terms of end to end delay of 66.61% for a period of 5 seconds and of 32.37% for a period of 20 seconds, and a gain over the protocol of [**?**] of 12.90% for a period of 5 seconds and of 13.87% for a period of 20 seconds.

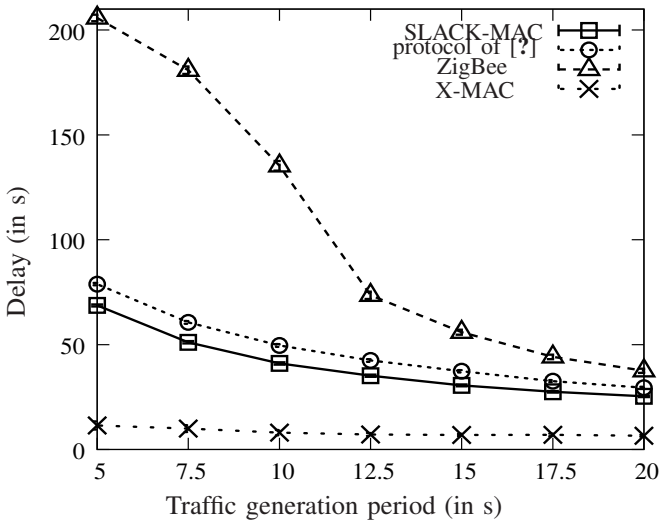

Figure 9. End-to-end delay of data packets as a function of the traffic generation period, for a duty-cycle of 1%, and for several protocols.

Figure **??** shows the duty-cycle in percent for each protocol. For ZigBee, the protocol of [**?**] and SLACK-MAC, the duty-cycle is set to 1% and is fixed. For X-MAC, the duty-cycle of node actually depends on the communication opportunities, as nodes having frames to send remain active until they can send their frames. Thus, X-MAC yields large duty-cycles, and consumes more energy than the other protocols.

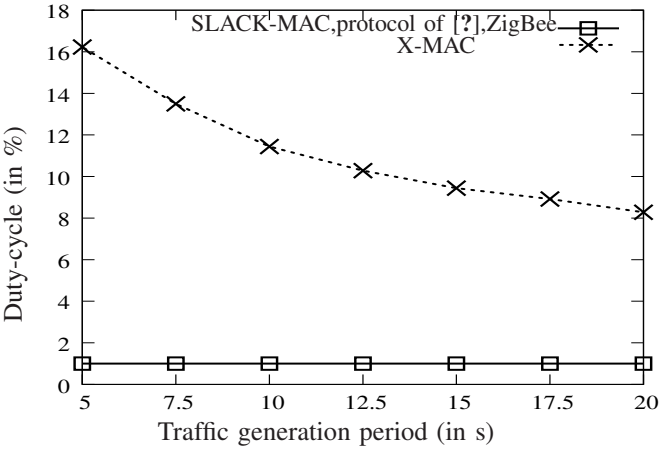

Figure 10. Duty-cycle (which is an approximation of the consumed energy) as a function of the traffic generation period, for a duty-cycle of 1%, and for several protocols.

Table **??** summarizes the gain of the best protocol over the other protocols, for three metrics. It can be seen that the gain of SLACK-MAC over all the other protocols is large, except for the end-to-end delay. For this metric, X-MAC has the lowest end-to-end delay, but this comes at the cost of a much larger energy consumption.

Table II
SUMMARY OF RESULTS

Metric	Best protocol	Gain over Zig-Bee	Gain over X-MAC	Gain over [?]	Gain over SLACK-MAC
Delivery ratio (%)	SLACK-MAC	79.31	19.90	1.04	-
Delay (%)	X-MAC	91.35	-	82.75	79.58
Duty-cycle (%)	SLACK-MAC, ZigBee, [?]	-	91.03	-	-

V. Conclusion

In this paper, we proposed the SLACK-MAC protocol for WSNs with low duty-cycles of 1%. Initially, nodes in SLACK-MAC activate their radio module randomly and independently, and build a history of successful communications. The history is used to determine the next activation times, which results into a self-adaptive behavior. We show that SLACK-MAC reaches a good behavior with a limited history size of only six entries. Only few activity cycles are needed to fill the memory lists. Then, we compare SLACK-MAC with existing protocols in terms of frame loss, end-to-end delay and consumed energy. We show that our low-cost protocol is able to significantly improve the performance of existing protocols. As future work, we aim to see if using an history can be applied to other existing probabilistic MAC protocols.

Acknowledgment

This research was partially supported by the “Digital Trust” Chair from the University of Auvergne Foundation.