Node Synchronization for Minimizing Delay and Energy Consumption in Low-Power-Listening MAC Protocols

Christophe J. Merlin and Wendi B. Heinzelman
Department of Electrical and Computer Engineering,
University of Rochester, Rochester NY
{merlin, wheinzel}@ece.rochester.edu

Abstract

Low-power-listening MAC protocols were designed to reduce idle listening, a major source of energy consumption in energy starved wireless sensor networks. Low-power-listening is a MAC strategy that allows nodes to sleep for t_i s (the "inter-listening" time) when there is no activity concerning them. It follows that a node has to occupy the medium for at least t_i s to guarantee that its destination will probe the channel at some point during the transmission. Low-power-listening protocols have evolved with the introduction of new radios, and the most recent contributions propose to interrupt communication between the sender and the receiver after the data packet has been successfully received and acknowledged. This results in significant energy savings because a sending node does not need to send for full t_i periods. We propose a new and simple approach to synchronize nodes on a slowly changing routing tree so that energy consumption is further reduced at the sending node, and the delay is considerably less. Our method allows the nodes to use a lower duty cycle, at no cost of overhead in most cases. Simulation and implementation results show that energy consumption can be reduced by a significant factor (dependant on t_i) and delay by at least 18%.

1 Introduction

Because it sits so low in the protocol stack, the MAC layer offers great potential to reduce energy consumption and delay. In very energy and bandwidth constrained wireless sensor networks (WSNs), every communication taking place between nodes has to happen according to the MAC protocol schedule. This transmission / reception schedule can be an important drain of energy. In simple CSMA MAC protocols like IEEE 802.11, a node spends most of the time in idle listening mode, a state in which the radio is turned on but not receiving any packet. This consumes as much energy as if the radio was receiving data. Several strategies have been employed to schedule packet transmissions differently, quickly dismissing IEEE 802.11 as unfit for WSNs. *Low-power-listening* (LPL) protocols take a different approach and aim to minimize idle listening, the main source of energy consumption in lightly loaded networks.

In a LPL protocol, nodes probe the channel every t_i s: they either stay on if they detect activity, or return to sleep for t_i s otherwise. Aloha with preamble sampling (PS) [3], WiseMAC [4], and B-MAC [10] were among the first LPL protocols to be proposed. Aloha-PS and B-MAC schedule packets to be sent with very long preambles of at least t_i s so that their destination wakes up sometime during the transmission. WiseMAC allows nodes to exchange active / sleep schedules in order to use shorter preambles. However, these protocols suffer from two main issues: in practice, they are not adapted to recent radios like the IEEE 802.15.4 [5] compliant Chipcon CC2420 [2] radio because the IEEE 802.15.4 standard has a fixed preamble length of only a few bytes. More importantly, their MAC schedules force receiving nodes to remain on for the duration of the preamble (a particularly costly situation since on the CC2420 radio, *Rx* mode drains more energy than *Tx* mode for any power setting). As a consequence, researchers introduced new LPL protocols whose schedules are compatible with the IEEE 802.15.4 compliant radios. X-MAC [1], SpeckMac-D [11], and MX-MAC [7] are among the most popular contributions. These protocols are based on repeating either the data packet itself (SpeckMAC-D and MX-MAC), or an advertisement packet (X-MAC), in place of a long preamble. The details of the transmission schedules (hereafter "MAC schedule") are given in Figure 1.

While the LPL family of MAC protocols generally lowers energy consumption without resorting to explicit exchange of active / inactive schedules between nodes, low duty cycles (or equivalently, high t_i values) drastically favor receiving nodes over mostly-sending nodes [7] and induce higher delays and contention. As Figure 1 shows, only one data packet can be transmitted per t_i cycle, which can

978-1-4244-2575-4/08/$20.00

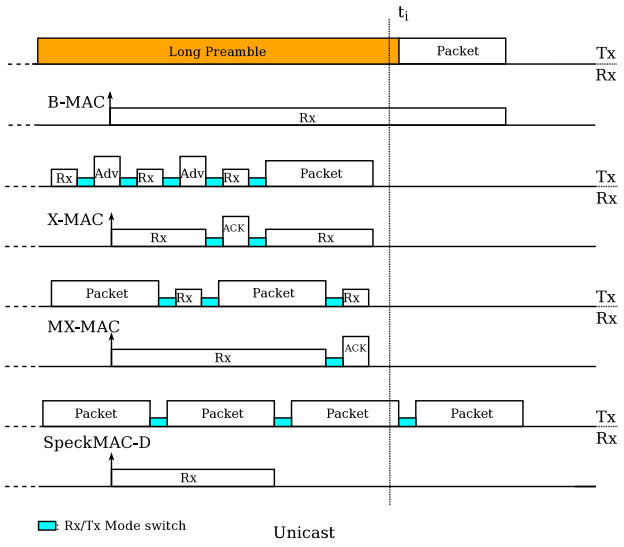


Figure 1. MAC schedule for B-MAC, X-MAC, MX-MAC, and SpeckMAC-D.

cause a packet to experience high delay over several hops, and the network to deliver small data rates. Concern for delay may force network designers to select a high duty cycle that would limit energy savings.

In this paper, we assume that all nodes share the same fixed t_i value, an assumption made by most LPL MAC protocols. Further, we propose to synchronize nodes along a slowly-changing routing path so as to minimize energy consumption and packet delay, *without* explicit scheduling between nodes or overhead of any sort. Only three LPL protocols can be selected to synchronize on unicast packets: X-MAC, C-MAC and MX-MAC. These protocols form a subfamily of LPL protocols that can be interrupted by the receiver. For unicast packets, the sender stops its stream of advertisement (X-MAC / C-MAC) or data (MX-MAC) packets after receiving an acknowledgement frame. Sender and receiver can then be synchronized to wake-up sequentially within a short interval. Conversely, SpeckMAC-D, which cannot be interrupted, needs explicit notification within nodes to synchronize.

We prove the benefits of node synchronization on packet rate, energy consumption and packet delay through simulation of an accurate model and implementation on a Tmote Sky [9] platform.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 explains the principle behind node synchronization for reduced delay and energy consumption. Sections 4 and 5 give simulation and implementation results that prove the concept and feasibility. Section 6 concludes our work and provides objectives for future work.

2 Related Work

WiseMAC [4] is a MAC protocol related to, but not directly part of the LPL family, as it relies on time synchronization to minimize the length of its preamble. Nodes running WiseMAC must exchange scheduling information so that a node with packets to send can start transmitting a short time before its intended receiver wakes up. In this study, we exclude protocols that cannot be implemented on newer radios and that require hard-to-achieve fine time synchronization. We show that explicit scheduling between nodes is unnecessary because it can be achieved implicitly with X-MAC, C-MAC, or MX-MAC.

Aloha with Preamble Sampling (PS) was one of the first channel probing schemes proposed for wireless sensor networks [3]. The MAC schedule of Aloha-PS resembles that of B-MAC in Figure 1.

B-MAC [10] with Low-Power-Listening (LPL) was the first MAC protocol to introduce channel probing schedules for recent radios. Polastre et al. provide a model for LPL with strong consideration for the target radio. To curb limitations imposed on the receiving node to stay awake for the time of the preamble, Polastre et al. propose sending packets with half-sized preambles.

Post-B-MAC protocols include SpeckMAC-D [11], X-MAC [1], and MX-MAC [7]. In [11], Wong and Arvind propose SpeckMAC-D, whose schedule replaces the long preamble with a stream of repeated data packets. Wong and Arvind develop a model for SpeckMAC protocols and study their impact on the ProSpeckz platform (a larger speck of size one square inch), while comparing them to B-MAC.

X-MAC replaces the long preamble of B-MAC by a stream of short advertisement packets. Upon hearing an advertisement packet containing information about the destination of the data packet, a node either returns to sleep or stays on to receive the data packet. There are several obvious limitations to the X-MAC schedule, including the risk for false acknowledgement since ACK frames are sent prior to receiving the data packet. Another significant drawback is that X-MAC is ill-adapted to broadcast packets: receiving nodes need to remain active until the end of the t_i period in order to receive the data packet. We showed in previous work [7] that there is another significant design issue with X-MAC: advertisement packets, which were meant to be very short, are difficult to hear. Because they are so small, clear channel assessments may be performed by the radio at intervals that miss the advertisement packets. This leads to error-prone wireless links.

We introduced MX-MAC in [7] as a response to the few design flaws of X-MAC. MX-MAC simply repeats the data packet instead of sending a stream of advertisement pack-

ets, and listens for ACK frames between packet transmissions. MX-MAC has thus a lower chance of false acknowledgments. Most importantly, MX-MAC is well suited for broadcast packets as nodes can receive data packets as soon as they hear an ongoing transmission. As our work showed, MX-MAC does not always outperform X-MAC, although it broadly allows for significant energy savings in broadcast and unicast mode. In this paper, we select MX-MAC over X-MAC, even though our results can be applied to X-MAC / C-MAC.

Finally, Lu et al. proposed DMAC [6], a MAC protocol whose goal, much like ours, is to stagger wake-up schedules over paths of a data-gathering tree. DMAC defines receive and transmit slots for unicast packet exchange at every node. In order to achieve synchronization, the slots are staggered along paths through the explicit exchange of schedules among neighbors. Because a node must know the Rx / Tx slots of its neighbors, DMAC requires local time synchronization. Additionally, broadcast packets from the data sink to the leaf nodes are only supported in specific slots, which may cause increased latency and energy waste when these slots are not used. Our approach conserves synchronization for these centrifugal flows. The improvements obtained by DMAC are significant and convincing, and we faced many of the same hurdles as Lu et al. However, our scheme achieves similar results *without* the overhead and limitations supposed by DMAC and comes at specifically no additional cost when the already popular X-MAC or MX-MAC LPL protocols are used within the scope of applications chosen for D-MAC.

3 Node Synchronization

In the following, the term "interruptible LPL" or *int-LPL* refers to the subfamily of LPL MAC protocols whose stream of packets can be interrupted by an acknowledgement frame; to the best of our knowledge, these are limited to the three MAC protocols X-MAC, C-MAC, and MX-MAC.

We chose to study only the MX-MAC protocol, although the results in this paper can be easily extended to the whole family of *int-LPL* protocols. Unlike X-MAC / C-MAC, MX-MAC is equally adapted to unicast and broadcast packets, and risks of false acknowledgement are smaller with MX-MAC. Most importantly, our previous work showed that the advertisement packets in X-MAC can be hard to hear [7], leading to rather poor link quality. Since our study applies to routing trees with at least two hops, the chance of packet delivery failure over one of the many hops on the routing path would be prohibitively high with X-MAC. We thus selected MX-MAC with 50B packets. For this data packet size, the packet delivery ratio over one hop is close to 98%, which translates into a packet failure rate of about

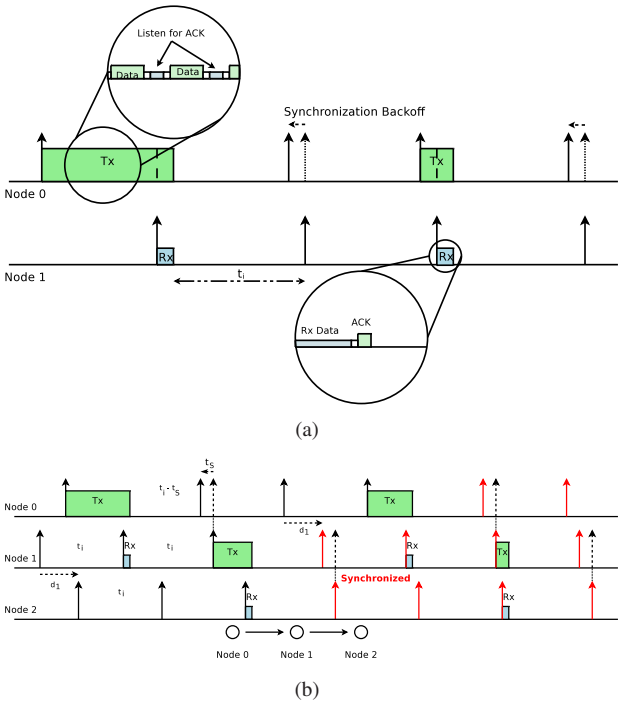


Figure 2. Synchronization principle for (a) two (b) three nodes running an *int-LPL* protocol.

92% over a four-hop path. For simplicity purposes, packets are delivered in a best-effort manner, and unsuccessful transmissions result in dropping the packet.

3.1 Synchronization Over a Unidirectional MX-MAC Link

3.1.1 Principle

Under the MX-MAC schedule, a node may learn of the active schedule of its destination when it receives an ACK frame after successfully transmitting a data packet. It is this particularity that allows nodes running MX-MAC to synchronize.

Consider two nodes 0 and 1, with a unidirectional link from 0 to 1. After receiving a packet, node 1 sets its timer to wake up t_i s later. The sending node 0 will do the same, although it will reduce its sleeping time by a small *synchronization back-off* $t_S > 0$. For the synchronization to take place, t_S must be greater than t_{Rx}, the time to receive a packet. This allows node 0 to wake up slightly before node 1 during the next rounds. Figure 2(a) shows how the synchronization takes place for two nodes.

The requirement of unidirectionality is a minor one: WSNs are usually characterized by centrifugal broadcast

packets (from the Data Sink to the peripheral nodes) and centripetal unicast packets (from the nodes to the Data Sink). Broadcast packets are commonly used to establish routes, refresh information about the end application, etc. On the other hand, unicast packets tend to flow from the periphery of the network to the data base. For the nodes to correctly synchronize, the unicast packets must follow a slowly-changing route. Moreover, regardless of the direction taken by broadcast packets, the schedule for broadcasting packets under MX-MAC does not break the existing synchronization between nodes as the broadcast schedule may not be interrupted by an ACK frame.

The synchronization process for more nodes is less intuitive. Synchronization over multiple hops is achieved by following the same rules: a sender must always back-off by the same amount of time after it has successfully sent a packet. For the case of three nodes, full route synchronization is not achieved until after two packets have been sent, as illustrated in Figure 2(b).

3.1.2 Synchronization Process

The number of unicast packets required to synchronize all nodes over a temporarily fixed routing path is a function of the number of hops. This observation, confirmed by preliminary results, can be modeled as follows.

Let $n = h$ be the number of hops from node 0 to node n. τ_i designates the relative time at which node i wakes up to probe the medium or send a packet. d_i separates τ_i and τ_{i+1} such that $\tau_i \leq \tau_{i+1}$. We have:

$$\tau_i = \tau_{i-1} + d_{i-1}$$
$$\tau_n = \tau_0 + \sum_{k=0}^{n-1} d_k$$

When node 0 sends the first packet to node 1, both nodes synchronize and their wake-up times differ by the synchronization time t_S:

$$\tau_1 = \tau_0 + t_S$$
$$\tau_n = \tau_0 + \sum_{k=1}^{n-1} d_k + t_S$$

At the i^{th} hop along the path, the transformation f happens:

$$\begin{cases} \tau_i & = \tau_{i-1} + t_S \\ \tau_{i+1} & = \tau_i + d_i \end{cases}$$
$$f(\tau_i) = \tau_{i-1} + t_S + d_i - t_S = \tau_{i-1} + d_i$$

At the last hop, node $n-1$ and n are separated by t_S:

$$\tau_n = \tau_{n-1} + t_S = \tau_{n-2} + d_{n-1} + t_S$$
$$= \tau_0 + \sum_{k=1}^{n-1} d_k + t_S$$

In other words, $\tau_n - \tau_0$ remains the same during the course of the first packet transmission from the source to the destination.

When the j^{th} packet is transmitted from node $i-1$ to node i we have:

$$\begin{cases} \tau_i & = \tau_{i-1} + t_S \\ \tau_{i+1} & = \tau_i + d_{i+j-1} \end{cases}$$
$$f^j(\tau_i) = \tau_{i-1} + t_S + d_{i+j-1} - t_S = \tau_{i-1} + d_{i+j-1}$$

Thus, after the j^{th} packet, we have:

$$\tau_n = \tau_0 + \sum_{k=j}^{n-1} d_k + jt_S$$

The nodes are synchronized when $\tau_n = \tau_0 + nt_S$, that is after at most $j = n$ packets have been properly sent[1].

Once the path is synchronized, the delay can be expected to be equal to $t_S + (n-1)(t_i + t_S) + t_{Rx}$, as suggested by Figure 3.

This short analysis also shows that clock drift has no effect on path synchronization because this process uses only the nodes' relative—not absolute—positions in time. Synchronization is reinforced with every packet sent, making this protocol resilient as long as the clock drift is significantly smaller than t_S, which can be expected.

3.1.3 Urgent Packets

Regular packets are forwarded in the next duty cycle after they have been received. On the other hand, urgent packets can be retransmitted immediately after they have been received. If a packet is marked as urgent (the implementation details are not relevant to, and beyond the scope of, this work) because of application or QoS requirements, the radio is kept on, waiting for the upper layers of the protocol stack to request sending the packet. The delay associated with urgent packets is less than t_i s, thus greatly reducing the packet delivery latency over regular packets. Over synchronized paths, the delay of urgent packets is equal to $nt_S + t_{Rx}$.

[1] Synchronization will happen as long as the j^{th} packet reaches at least node $n - (j-1)$

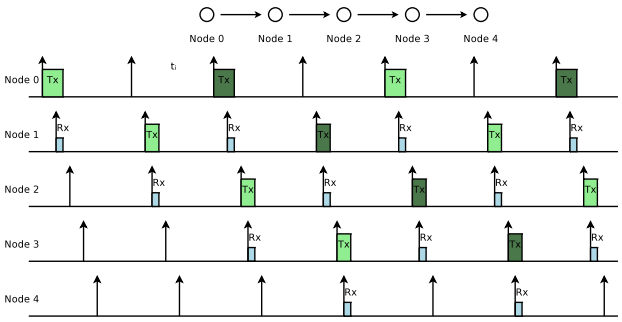


Figure 3. Node 0 pipelines packets and increases the packet rate.

The decision to send urgent packets within the same t_i period, but to exclude regular packets from immediate retransmission is a design choice motivated by practical implementation considerations. Support for urgent packets requires protocols from the Data Link layer to the Routing Protocol to collaborate and capably handle urgent deliveries. Today, this is rarely the case. Processing on each packet (snooping, queue reordering, next-hop calculation, loading the radio FIFO, etc.) must be very limited in order to meet the next-hop's wake-up time. If t_p is the processing time, we must have $t_{Rx} + t_p < t_S$.

Moreover, nodes sending urgent packets would be at a disadvantage when competing with a neighbor that is not retransmitting packets, but originating them. Such a neighbor would consistently send packets $t_{Rx} + t_p$ s before the node, and while rare, this configuration would deny any use of the channel if the neighbor created a packet every t_i s.

In spite of these caveats, a protocol designer may wish to treat all packets as urgent ones, and would thus benefit from very short delays.

3.1.4 Pipelining of Packets on a Synchronized Path

Because packet transmissions happen in a sequential way, packets can be pipelined over the path so that a packet is sent every $2t_i$, as illustrated by Figure 3.

Pipelining is only possible with synchronized nodes because if nodes are not synchronized they would interfere with one another and exacerbate the hidden node problem, common to all LPL protocols.

3.2 Synchronization Over Several Unidirectional Paths and Conflict Resolution

In some specific cases, the risk for packet collision still exists on synchronized paths. This is particularly true when a routing tree is formed of two or more parallel branches:

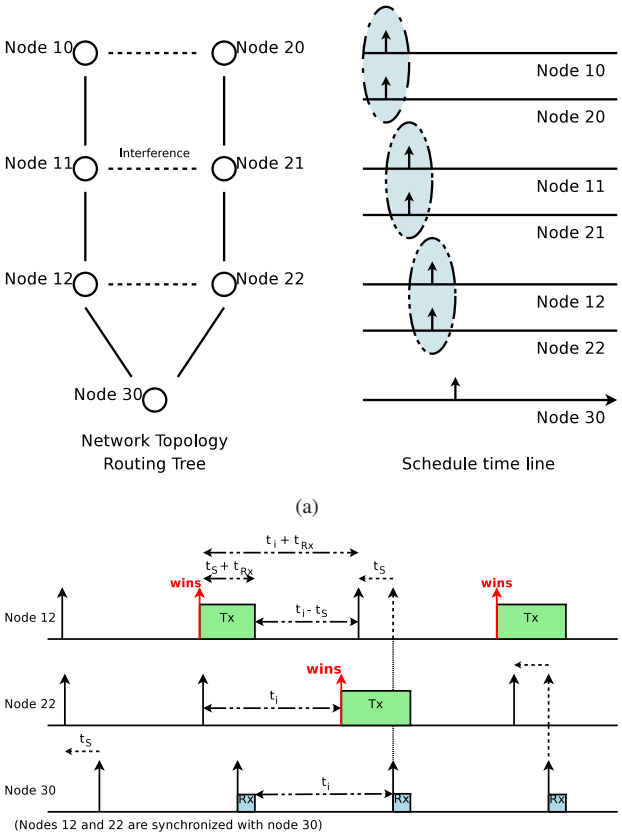


Figure 4. (a) Synchronized nodes along two parallel paths: nodes $\{10, 11, 12, 30\}$ form one path, and $\{20, 21, 22, 30\}$ another one. The dotted lines indicate that the nodes can communicate with each other (and thus interfere). (b) Mitigation of the problem.

nodes i hops away from the destination tend to wake up at the same time, causing contention. This node configuration is illustrated by Figure 4(a).

The incidence of this problem depends on several factors such as the routing protocol (which may forward packets along parallel paths for robustness), the network topology (nodes from the same region may report highly redundant information if no packet fusion strategy is employed) and the application (which may require high data rates from collocated sources).

Several techniques may be used to mitigate this phenomenon, including information exchange among neighbors, packet rate reduction, etc. However, the node schedule already offers a good solution to prevent collisions and to guarantee fairness among information flows. If two neighboring nodes are part of two different synchronized paths

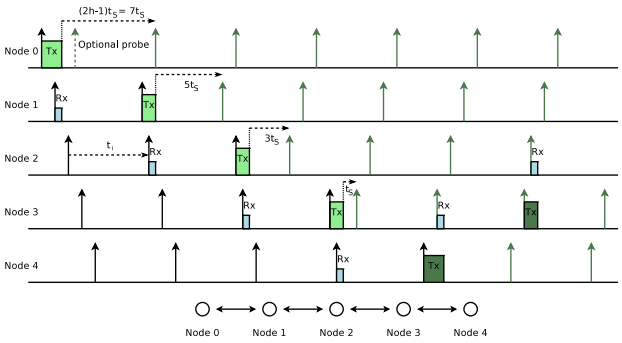


Figure 5. Bidirectional path synchronization time line.

as Nodes 12 and 22 are in Figure 4(a), they will attempt to send packets at about the same time. However, if node 12 can send its packet, it will wake up slightly after node 22. This is because node 12 will back off by t_S from the moment it receives an ACK frame, which happens after the time it takes to receive the data packet (t_{Rx})—a few tens of milliseconds. Figure 4(b) illustrates this with a time line: after sending a packet, nodes 12 and 22 are separated by t_{Rx} s. They alternatively wake up before the other one as they send packets, guaranteeing fairness between the two branches of the routing tree.

3.3 Synchronization Over a Bidirectional Path

Although uncommon in WSNs, some network topologies and applications may send unicast packets over paths that are in part or in whole bidirectional. This could be the case when several data sinks are deployed in the network. We developed an algorithm that coordinates bidirectionality on a path, although it induces overhead.

Let nodes i and j be the two ends of a sub-path $P_{i\to j}$. In order to synchronize nodes over $P_{i\leftrightarrow j}$, the MAC protocol must allow packets to travel in only one direction at a time during synchronized rounds. Furthermore, cross-layer information such as the number of packets sent per round, and the number of hops from i and j is needed.

Upon forwarding the last packet of the synchronized round, each node backs-off by $-t_S(2h_{k,\{i,j\}} - 1)$ s, where $h_{k,\{i,j\}}$ is the number of hops from node k to i or j. For space considerations, and because this case is the exception rather than the norm, we do not explain the bidirectional synchronization algorithm in detail. However, it is available at *www.ece.rochester.edu/∼merlin/NodeSynchronization/NodeSyncURTR.pdf* and we provide Figure 5 to illustrate this bidirectional synchronization process.

4 Simulation Results

In this section, we explore the advantages and limits of node synchronization through Matlab simulations. Following our work on MAC protocols in [7], we developed a very accurate Matlab model for time and energy consumption of the Tmote Sky [9] platform running MX-MAC. Through a data acquisition board, we very precisely measured the time and energy expended by each basic operation of the MX-MAC protocol, including channel probe and packet reception. We then created a Matlab program that reconstructs a desired scenario and estimates the energy use over the simulation time. We verified our reconstruction model and found it to be within 3% of the actual energy consumption. Through practical experience, we were able to shed preconceived ideas about radio and MAC behaviors. Thus, the results provided by this section are those of an implementation reconstruction, rather than those of a simulation. However, to distinguish between direct results from our implementation, we use the words *reconstruction* or *simulation*.

In this section, 10 nodes are randomly placed to form a multi-hop network. Unless otherwise specified, a source node sends packets at a rate of ${}^1/_2 ptk.s^{-1}$.

4.1 Synchronization Principle

We simulated a scenario in which $t_i = 1.5$ s, and the synchronization back-off t_S is 50 ms. Notice that the t_S value is about twice the reception time of a 50B packet. The duty cycle was chosen to be fairly low, since we expect that the improvements brought by path synchronization will allow the t_i value to increase.

Figure 6 shows that 5 nodes synchronize on the temporarily fixed path {1, 5, 4, 3, 10}. In the Figure, a triangular marker $\triangle$ represents a probe, $\circ$ a packet to send, $*$ a successful packet reception and $\times$ a failed one. After one packet, nodes 3 and 10 have staggered probes, nodes 4, 3 and 10 after the second packet.

This scenario also illustrates the behavior of the nodes when synchrony is lost: in the worst case scenario, it would take n packets to reconstruct a synchronized path. However, complete loss of synchronization is highly unusual because whenever a node fails to receive a packet from its neighbor, it simply wakes up t_i seconds later, *i.e.*, in the same relative time position.

4.2 Packet Delay

Next, we investigate the packet delay after the nodes have been correctly synchronized. We define packet delay as the time between the first attempt to send a packet and the successful reception of this packet, noting however that the packet could have been created at most t_i s before the first

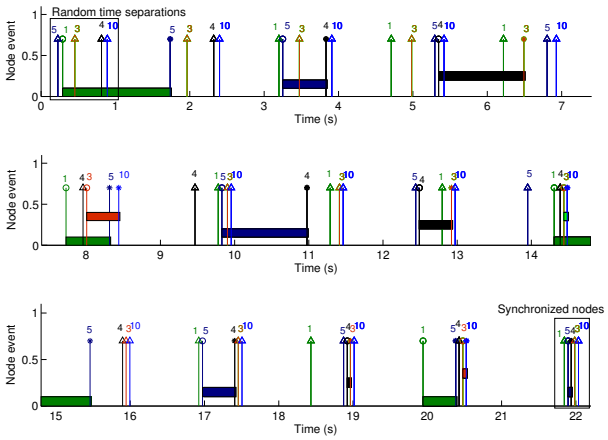


Figure 6. On the path {1, 5, 4, 3, 10}, the nodes synchronize correctly after only a few packets.

transmission attempt. We consider that if a synchronized path is incapable of transmitting the required packet rate (the node queue keeps expanding), t_i needs to be lowered in order to accommodate higher traffic. We offer a solution to do so in [8].

4.2.1 Delay of Non-Urgent Packets

Figure 7(a) shows the packet delay for the node configuration of the previous section. Because of the initial time differences between node schedules, the path was synchronized in only 4 packets. The packet delay then hovers around $4.74\ s$, which is approximately equal to $t_S + 3(t_i + t_S) + t_{Rx}$ (t_{Rx} is modeled by a random variable with a normal distribution).

The first packet is sent without any synchronization between the nodes and its delay is $5.8\ s$, a value that depends on the initial random wake-up times. Synchronization cut the packet delay by over 18%.

4.2.2 Packet Delay of Urgent Packets

Figure 7(b) shows the packet delay of non-urgent and urgent packets. The very first packet is sent without any synchronization, and takes $7.47\ s$ to be delivered. Once synchronized, the delay is reduced by 35%. Packets 7 through 15 are marked as urgent: they are delivered almost immediately, with a delay of around $220\ ms$, which corresponds to $nt_S + t_{Rx}$ when $n = 4$. The simulation shows that the synchronization is not broken by urgent packets.

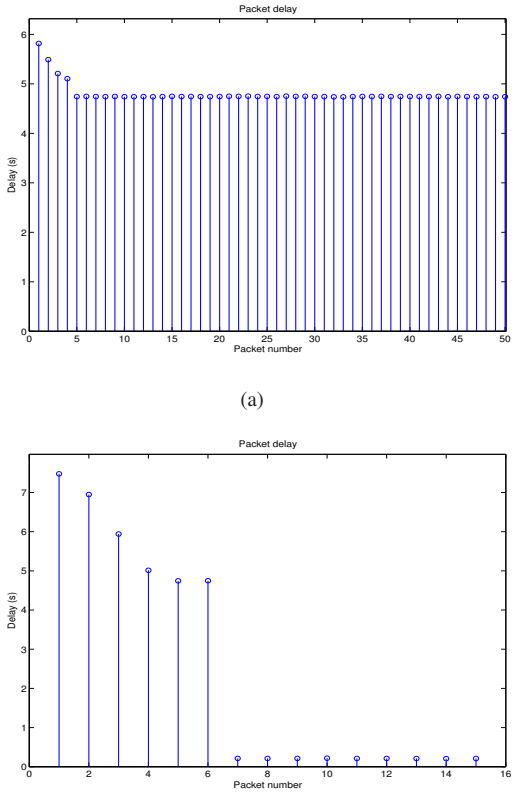


Figure 7. (a) Packet delay on the same path as Figure 6. (b) Same configuration, but packets 7 through 15 are marked as urgent.

4.3 Bidirectionality

Although it is rarely expected in a WSN, bidirectionality may exist. We tested our algorithm over the same five-hop, now bidirectional, path {1, 5, 4, 3, 10}. We had to lower the t_i value to $0.5\ s$ in order to accommodate a larger load on the path. The larger issue of t_i control for *int-LPL* is addressed in [8], which offers a method inspired by control theory to dynamically set the duty cycle. In this section, the packet delay is defined as the time difference between the moment a packet is intended for delivery and the time when it is successfully received. This definition, slightly different from the other cases, allows results to reflect the time spent in the queue by a packet.

Figure 8 shows the delay for bidirectional packets sent by node 1 at a rate of $^1/_3\ pkt.s^{-1}$ and node 10 at $^1/_5\ pkt.s^{-1}$. The average packet delay comes at $3.6\ s$, mostly because packets must be queued while a node is not allowed to use the path. The very first unidirectional packet experiences a high delay ($4\ s$), and the following two must be queued

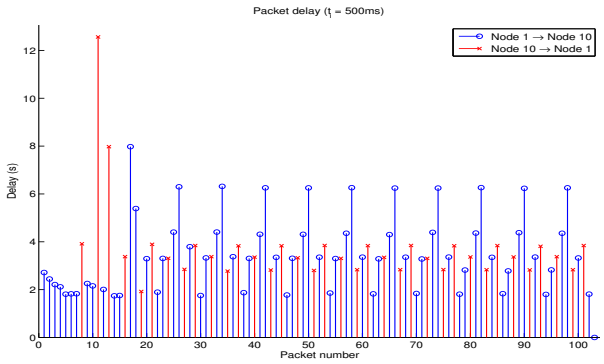


Figure 8. Delay on the same path as Figure 6 for bidirectional packets.

while the bidirectional path is established. Note that the packet transmission from node 1 to 10 occupies the channel for the same amount of time as unidirectional packets (because the various nodes are synchronized). We can then consider that, on average, packets spend $1.8\ s$ in the queue for $1.8\ s$ of travel from node 1 to 10.

This shows that our algorithm is capable of maintaining synchronization over a bidirectional path, while keeping packet delay in check.

4.4 Energy Consumption

In order to fairly evaluate the energy benefit of node synchronization (and not just of LPL schemes), we compared the energy consumption of the proposed scheme with that of nodes running MX-MAC where neighbors wake up randomly within the t_i time interval.

Table 1 gives the average energy consumption of nodes on a path. Node 0 is furthest from node $n = h$, the destination. With our naming convention, h is also the maximum number of hops on the path. Because the destination node h is always receiving, its energy consumption is very low and depends only on the t_i value (as t_i increases, node h still uses the same amount of energy to receive packets, but it performs fewer channel probes). Thus, we excluded the energy consumption of node h from the average.

Typically, the non-synchronized nodes consume on average 6 times as much energy as the synchronized case. The reason for this difference is given by the average delay shown in the Table. When $h = 1$, the packet delay is about $^{t_i}/_2 \approx 0.76\ s$ when the nodes are not synchronized, and $t_S + t_{Rx} \approx 65\ ms$ otherwise. This means that non-synchronized nodes must spend much more time with their radio active and transmitting than synchronized nodes. Consequently, the per-node average energy consumption greatly increases, by a factor of about $^{t_i}/_{2(t_S+t_{Rx})}$.

Table 1. Energy Consumption and Packet Delay.

Parameter		Sync		Non-sync	
		Av. Energy (J)	Av. Delay (s)	Av. Energy (J)	Av. Delay (s)
h ($t_i = 1.5$)	1	1.59	0.065	12.73	0.76
	2	1.50	1.63	9.76	2.91
	3	1.49	3.21	9.37	6.10
	4	1.54	4.91	9.00	10.63
	5	1.45	6.34	8.80	13.54
t_i ($h = 4$)	0.5	1.77	1.74	5.73	3.08
	1	1.67	3.26	8.99	6.62
	1.5	1.54	4.91	9.00	10.63
	2	1.31	6.26	9.10	13.98

When the number of hops h is fixed and equal to 4, an increase in t_i reduces the per-node average energy consumption. This is only true when nodes are synchronized: if t_i increases, non-synchronized nodes must spend more time transmitting (for $^{t_i}/_2\ s$ on average). On the other hand, synchronized nodes must send for approximately $t_S + t_{Rx}\ s$, whatever the duty cycle. However, as the duty cycle decreases, the nodes have to spend less energy probing the medium, and thus synchronized nodes see their global energy consumption reduced.

5 Implementation Results

We implemented the principles behind node synchronization in TinyOS for the Tmote Sky platform. This includes code for MX-MAC, as well as for the back-off techniques described here. We present results from this implementation.

5.1 Methodology

Once the MX-MAC code was set onto the Tmotes, gathering results about packet delays appeared to be an intractable issue. In order to visualize synchronization, we needed to deploy a network of more than one hop. We chose to replicate the case of $h = 4$ (in a linear topology), often used in our simulations. In order to demonstrate synchronization, we opted to let Matlab—not the motes themselves—collect information about the packets. This is because in very time-sensitive MX-MAC, time stamping operations can be a delicate task for which CPU resources may not always be available on the motes. This however meant that all motes had to be in range of one-another and of the computer running Matlab, and had to be loaded with predefined neighbor graphs.

Yet, with this solution, motes that in a real deployment would not have to compete for the channel could now hear each other. However, because of the nature of synchronized

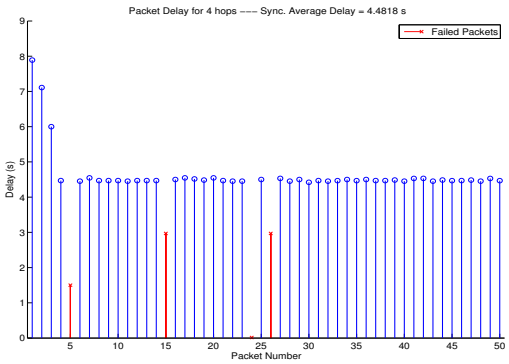


Figure 9. Packet delay over the same path of the implementation (failed packets do not reach the eventual destination and are not counted in the average delay).

paths, packet collisions from motes placed at different levels of the routing tree did not compete for the medium at the same time, thus considerably alleviating this problem.

Our results are obtained from a mote receiving all packets transmitted over the channel and forwarding them to Matlab. Consequently, we cannot display channel probes since they are "silent" (the radio is in Rx mode only). We present results in spite of these caveats.

Finally, we cannot show the energy consumption of our implementation. This is because the Tmote sky can only measure its internal voltage through the ADC. This value is typically noisy, and the battery voltage does not evolve as a linear function of the energy remaining. Because the MX-MAC protocol is very energy efficient, the voltage drop over a measurable period of time is well within the natural ADC noise, with or without node synchronization.

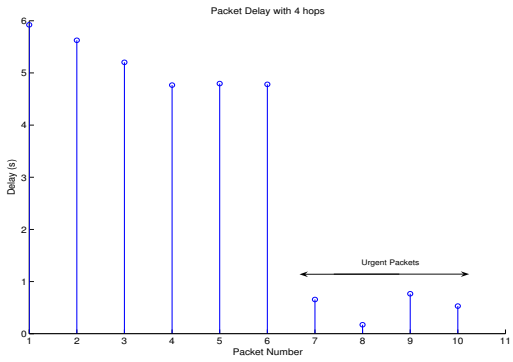
5.2 Synchronization Principle

The goal of this section is to prove that node synchronization is practical and offers results in actual platform implementations.

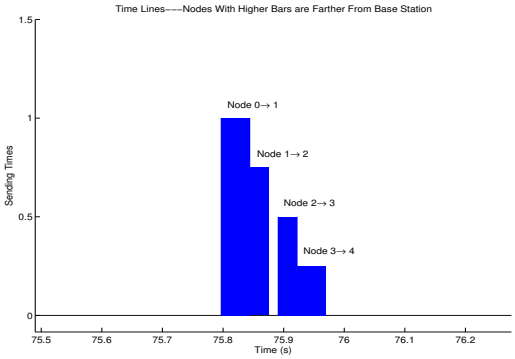
Figure 9 shows that the motes on the 4 hop path $P_{0\to1\to2\to3\to4}$ successfully synchronize after the predicted number of packets. They also send packets in about $4.5\ s$ once they are synchronized, which is the delay predicted by the simulation model (within 4%, probably because Matlab starts time-stamping packets only *after* the first one has been received, and stops *before* the ACK frame is sent).

5.3 Urgent Packets

Next, we present the delay of urgent packets in Figure 10(a) and confirm the results obtained through the re-



(a)



(b)

Figure 10. (a) Delay of non-urgent and urgent packets over a synchronized path. (b) Time line of transmissions of an urgent packet.

construction model. Packets 7 through 10 are marked as urgent, and they see their delay hover between $766\ ms$ and $172\ ms$, the actual value of the delay being hard to measure because of the typically slow link between the mote and the PC. It also shows that urgent packets do not break the path synchronization.

The fast delivery of urgent packets is obtained through the immediate repetition of a received packet as shown in Figure 10(b). Small variations in the transmission times at every mote are natural, but the differences observed here are mostly due to the capture by Matlab, which does not receive ACK frames and can only deduce that a transmission has ended after another has started. This causes some bars in the graph to be "glued" together.

5.4 Packet Pipelining

Finally, Figure 11 shows the medium activity when the source node 0 sends a packet every $2t_i$. Thanks to node synchronization, packet transmissions can be staggered over

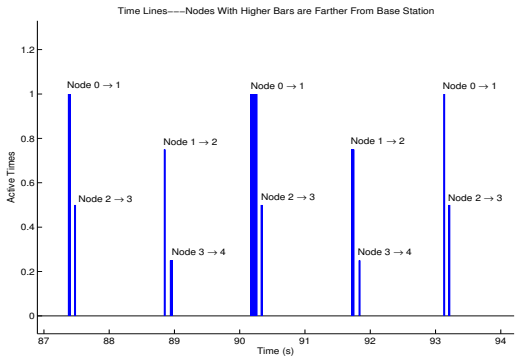


Figure 11. Packet pipelining: node 0 sends a new packet every $2t_i$. Transmissions end up being staggered.

non-continuous wireless links. The packet rate can then climb to ${}^{1}/_{2t_i}\ pkt.s^{-1}$, even though the packet delay remains the same.

This set of results shows that node synchronization over a temporarily fixed path is practical and works for the cases tested in simulation.

6 Conclusion and Future Work

The family of interruptible low-power-listening MAC protocols is a powerful set of protocols recently developed. Only now are we beginning to use them to the extent of their potential. We showed in previous work [7] that lower duty cycles tended to increase the energy consumption of mostly sending nodes, which, with packet delay, was an important obstacle to higher t_i values.

We now propose a simple approach to synchronizing nodes on a temporarily-fixed path. Through analysis, we proved that the path is automatically synchronized after $n = h$ packets have been sent from node 0 (the farthest) to node n. In other words, the requirement to have a fixed path is a weak one.

Node synchronization has several benefits: it drastically reduces the packet delay, and it reduces the energy use at every node by a factor of about ${}^{t_i}/_{2t_S}$, removing the limit standing in the way of lower duty cycles.

In addition, we proposed several strategies to increase the packet rate and further reduce the packet delay. By pipelining packets over synchronized paths, we doubled the packet rate. Our approach also allows urgent packets to be delivered almost immediately, taking the delay from $t_S + (n-1)(t_i + t_S) + t_{Rx}$ to $nt_S + t_{Rx}$.

Through simulation and implementation on the Tmote Sky platform with TinyOS, we showed that node synchronization is both possible and practical.

Maybe most importantly, the improvements on the node lifetime and packet delays require no overhead or cost in most WSN cases: nodes do not need to exchange On / Off schedules with their neighbors, and in the unidirectional case, no explicit synchronization phase or messages are required. Simply by the sheer MAC schedules used by MX-MAC and X-MAC / C-MAC can the nodes organize themselves automatically.

In future work, we plan to investigate further special node deployment cases, such as those that require bidirectionality. This will require developing cross-layer routing protocols and applications capable of taking advantage of the MAC schedules: for instance, it may be beneficial to let the application send a packet a little before the end of the t_i interval. We will also research issues of fairness further.

References

[1] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *Proc. 2^{nd} ACM Conf. on Embedded Networked Sensor Systems (SenSys'06)*, pages 307–320, 2006.

[2] Chipcon Products from Texas Instruments. CC2420 data sheet, 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver.

[3] A. El-Hoiydi. Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks. In *Proc. IEEE Int. Conf. on Communications (ICC)*, Apr. 2002.

[4] A. El-Hoiydi and J. Decotignie. WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks. In *Proc. 1^{st} Int. Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, 2004.

[5] IEEE Computer Society LAN MAN Standards Committee. Wireless medium access control (MAC) and physical layer (PHY) specifications for low rate wireless personal area networks (LR-WPANs). In *IEEE Std. 802.15*, 2004.

[6] G. Lu, B. Krishnamachari, and C. Raghavendra. An adaptive energy-efficient and low-latency MAC for data gathering in sensor networks. In *Proc. 4^{th} International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN)*, 2004.

[7] C. J. Merlin and W. B. Heinzelman. Network-aware adaptation of mac scheduling for wireless sensor networks. In *Proc. 3^{rd} Conf. on Distributed Computing in Sensor Systems (DCOSS'07 Poster Session)*, June 2007.

[8] C. J. Merlin and W. B. Heinzelman. Duty cycle control for low power listening mac protocols. In *Proc. MASS'08*, Sept. 2008.

[9] MoteIV Tmote sky. http://www.moteiv.com/tmote.

[10] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proc. SenSys'04*, pages 95–107, Nov. 2004.

[11] K.-J. Wong and D. Arvind. Speckmac: Low-power decentralised mac protocol low data rate transmissions in specknets. In *Proc. 2^{nd} IEEE Int. Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality (REALMAN'06)*, May 2006.