Schedule Adaptation of Low-Power-Listening Protocols for Wireless Sensor Networks

Christophe J. Merlin and Wendi B. Heinzelman
Department of Electrical and Computer Engineering,
University of Rochester, Rochester NY

***Abstract*—Many recent advances in MAC protocols for wireless sensor networks have been proposed to reduce idle listening, an energy wasteful state of the radio. Low-Power-Listening (*LPL*) protocols transmit packets for t_i s (the "inter-listening interval"), thereby allowing nodes to sleep for long periods of time between channel probes. The inter-listening interval as well as the particular type of *LPL* protocol should be well matched to the network conditions. In this paper, we propose network-aware adaptation of the specific succession of repeated packets over the t_i interval (the "MAC schedule"), which yields significant energy savings. Moreover, some *LPL* protocols interrupt communication between the sender and the receiver after the data packet has been successfully received. We propose a new and simple adaptation of the "transmit / receive schedule" to synchronize nodes on a slowly changing path so that energy consumption and delay are further reduced, at no cost of overhead in most cases. Our results show that using network-aware adaptation of the MAC schedule provides up to 30% increase in lifetime for different traffic scenarios. Additional adaptation of the transmit / receive schedule to automatically synchronize the nodes can reduce packet delivery delays by up to 50%, providing an additional decrease in energy consumption of 18%.**

***Index Terms*—Low-Power-Listening, MAC, Path Synchronization, Adaptation.**

I. Introduction

APPLICATIONS for wireless sensor networks (*WSN*) are becoming increasingly complex, and they require the network to maintain a satisfactory level of operation for extended periods of time. Consequently, sensor networks have to make the best possible use of their initial energy resources, specifically by constantly adapting their protocols to the changing conditions in the network. Both protocol-specific and cross-layer schemes have offered a plethora of energy reducing techniques. In particular, there are several protocols that focus on reducing energy at the data link / MAC layer, which constitutes the scope of this work. In this paper, we investigate how to keep the radio in its energy-conserving sleep mode for as long as possible.

We offer three ways to adapt several key aspects of MAC protocols. The first idea presented in this paper discusses switching between MAC schedules to adopt the most energy-efficient pattern of packet transmissions and receptions. Because different areas in the network experience different and changing loads of traffic, the MAC protocol should utilize the schedule most economical for the local conditions. Secondly, we propose to synchronize nodes so as to reduce transmission time and thus energy consumption and packet delivery delays. A third technique controls the inter-listening time to conditions in the network and is exposed in [1].

As new sensor network platforms have appeared on the market, a simple observation was made that idle listening, far from being negligible, was a major source of energy consumption [2] [3] [4]. Low-Power-Listening (*LPL*) and Preamble Sampling (PS) MAC protocols were introduced as a result. In his taxonomy of MAC protocols [5], Langendoën identifies *LPL* and *PS* protocols as two branches of random access MAC protocols, with the only difference that *LPL* MAC protocols need not know anything about their neighbors and their wake-up schedules. Both types of MAC protocols, including B-MAC [2], WiseMAC [6], SyncWUF [7] and X-MAC [8], use the insight behind Aloha with PS [9]: the sending node occupies the medium for long t_i[1] intervals to signal its imminent packet transmission. Receiving nodes are thus allowed to sleep for at most the duration of this preamble (t_i s), and they must stay awake when they sense a busy medium until the packet transfer is complete. In this work, we consider only the *LPL* branch of the Langendoën taxonomy (although many of our results can be transposed to other MAC protocols), and we define "(*LPL*) MAC schedule" as the pattern of packet transmissions occurring within the t_i interval.

Changes in radios have forced researchers to abandon B-MAC and a few other *LPL* protocols in some cases: although it paved the way to new MAC protocols, B-MAC, which uses a variable-length preamble to signal the impending packet transmission, can no longer be implemented as proposed on the new IEEE 802.15.4 compliant platforms because this standard has a fixed preamble length of only a few bytes. We assume such a target radio, and make design and research decisions accordingly—thus B-MAC is not included in our work. After the introduction of new radios, researchers introduced new *LPL* and *PS* protocols: X-MAC [8], C-MAC [10], WiseMAC [6], CSMA-MPS [11] and SpeckMac [12] are among the most popular contributions. These protocols are based on repeating either the data packet itself (SpeckMAC and CSMA-MPS), or an advertisement packet (X-MAC / C-MAC), in place of a long preamble. The details of the transmission schedules (the "MAC schedules") are given in Figure 1.

In the initial part of this work, we prove that while the *LPL* family of MAC protocols generally lowers energy consump-

This work was supported in part by NSF # CNS-0448046

[1]The notation t_i was borrowed from the work in [2].

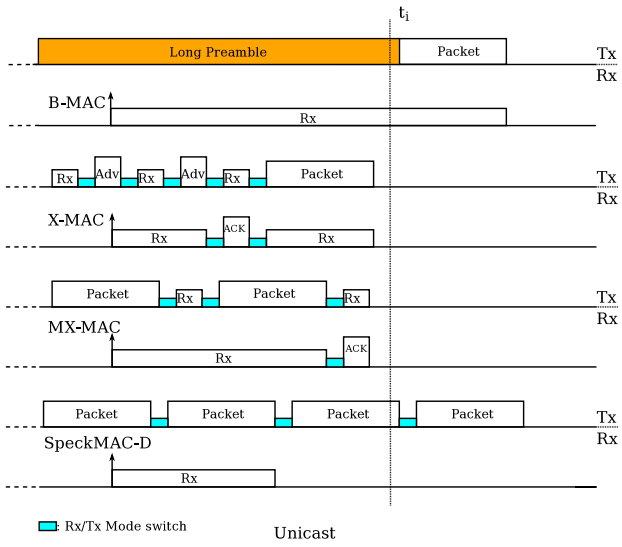


Fig. 1. MAC schedule for B-MAC, X-MAC, MX-MAC, and SpeckMAC.

tion 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 and induce higher delays and contention. As Figure 1 shows, for these *LPL* protocols, only one data packet can be transmitted per t_i cycle, which can 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. We address this problem in the second part of our work by synchronizing the transmitting / receiving schedules of nodes on a slowly-changing routing tree.

This paper's contributions are threefold:

- We propose switching MAC schedules from a pool of MAC protocols at the transmitter to minimize energy consumption based on parameters such as packet size, whether the packet is broadcast or unicast, and the estimated ratio of transmit to receive packets in the local neighborhood. The protocols are "compatible" because they are interchangeable: the receiver does not need to know what specific schedule is being used, it simply wakes up and senses the channel every t_i seconds and sends an ACK frame when required by the received packet. As a consequence, this protocol, called *MiX-MAC*, requires no overhead, and our implementation of this approach shows that lifetime gains can reach up to 30%.
- Because we utilize existing MAC protocols for our pool, we provide a detailed study of two existing *LPL* MAC protocols, X-MAC and SpeckMAC, in a head-to-head comparison, showing the advantages and disadvantages of each approach for both unicast and broadcast packets. We also identify MX-MAC, a modified version of CSMA-MPS, that is compatible with the X-MAC and SpeckMAC schedules.
- 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, which cannot be interrupted, needs explicit notification within nodes to synchronize.

This paper continues with a discussion of related work in Section II. Section III then introduces MiX-MAC, one of the main concepts of this work. Section IV provides details of our implementation of various MAC protocols on the Tmote Sky motes, as well as results from experiments with this implementation, showing the advantages of MiX-MAC. We introduce node synchronization, the second concept of this work, in Section V. Section VI provides simulation and implementation results showing the benefits of synchronizing transmit / receive schedules for the individual *LPL* protocols as well as for MiX-MAC. Finally, Section VII concludes this work.

II. Related Work

Aloha with Preamble Sampling (PS) was one of the first channel probing schemes proposed for wireless sensor networks [9]. In this approach, packets are sent with a preamble greater than or equal to the channel check interval t_i. Nodes periodically wake up and sense the medium. If the channel is busy, the probing node stays in receive mode until the data packet transmission is complete. Otherwise, the probing node goes back to sleep. El-Hoiydi developed an analytical model for Aloha with *PS* and studied its performance using four metrics: throughput, delay, power consumption, and lifetime. The transmit and receive powers assumed by El-Hoiydi in [9] led the author to recommend limiting the use of Aloha with PS.

Following Aloha with PS, El-Hoiydi et al. introduced WiseMAC [6], a MAC protocol that reduces the preamble length before sending a data packet by exchanging wake-up schedules between neighbors. However, WiseMAC as originally proposed (like B-MAC) cannot be implemented on 802.15.4 radios[2]. Moreover, piggybacking scheduling information to acknowledgment frames, as required by WiseMAC, supposes that hardware acknowledgments may not be used. Hardware ACK frames are considerably faster than software acknowledgments (by a factor of two to five depending on packet size and code optimization) and allow the radio to return to sleep much earlier, resulting in significant energy savings. For these reasons, and because it is classified under the *PS* family of MAC protocols, it was not included in most of our study. We show that explicit scheduling between nodes

[2]An updated version of WiseMAC was proposed for 802.15.4 compliant radios, where a repetition of frames, similar to SpeckMAC's schedule, replaced a long preamble.

is unnecessary because it can be achieved implicitly with X-MAC, C-MAC, or MX-MAC.

B-MAC [2] with *LPL* was the first MAC protocol to introduce *LPL* schedules for recent radios (with WiseMAC being the first for *PS* MAC protocols). Polastre et al. provide a model for *LPL* with strong consideration for the target radio. The authors thoroughly compare B-MAC to S-MAC [3] and T-MAC [4]. 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 X-MAC [8] and SpeckMAC [12]. Both protocols are of the channel-probing family and tried to improve the *LPL* scheme presented by B-MAC. Further explanation of these protocols is provided in Section III.

Although more recent, C-MAC [10] uses the same schedule as X-MAC and is therefore included in our work under the same principles that govern X-MAC.

In [12], Wong and Arvind propose SpeckMAC, a family of *LPL* MAC protocols, which include SpeckMAC-B and SpeckMAC-D. The SpeckMAC protocol family is intended for miniature motes called specks. SpeckMAC-B stands for *Back off* and replaces the long preamble with a sequence of wake up packets containing the destination target and the time when the data packet will be sent. This allows receiving nodes to sleep for the remainder of t_i and activate just in time for data reception. However, this scheduling supposes fine time synchronization between nodes, which we do not assume in this work. Wong and Arvind develop a model for the SpeckMAC protocols and study their impact on the ProSpeckz platform (a larger speck of size one square inch), while comparing them to B-MAC. SpeckMAC-D is described in Section III-D and Figure 1.

Keshavarzian et al. proposed to stagger wake-up schedules for random access protocols [13]. Several cases, depending on the direction (forward or backward) of the traffic, are considered. The "shifted even and odd" wake-up pattern reduces packet delivery delays: all nodes shift their wake-up schedule by $^T/_2$ with respect to their previous hop, where T is the inter-listening time. At worst, the delay is $(h+1)\frac{T}{2}$. The ladder pattern forces nodes to wake up only a few milliseconds after their previous hop. Multi-parent ladders account for the case when networks have multiple branches: the inter-wake-up time T is divided to accommodate various branches.

Similarly, Lu et al. proposed DMAC [14], 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 transmit / receive schedule synchronization 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 interruptible *LPL* protocols X-MAC, C-MAC or MX-MAC protocols are used within the scope of applications chosen for D-MAC.

Much work has been dedicated to the task of adapting MAC protocols to conditions in the local neighborhood of a node [4] [15] [16]. The authors in [15] propose several variants of 1-hopMAC, which is a receiver-based cross-layer routing and MAC protocol. Watteyne et al. propose switching between two different schedules based on a routing function f. MiX-MAC follows a similar idea, although it utilizes a wider range of parameters to compute switching thresholds. Moreover, MiX-MAC considers schedules for broadcast packets, which is not addressed in [15]. In [4], van Dam and Langendöen propose to improve S-MAC by a novel adaptive active / sleep duty cycle. Their protocol, T-MAC sends packets in bursts during active periods, and if no activity is detected during a small window of time, nodes return to sleep. As a consequence, nodes have a shorter duty cycle under T-MAC. In [16], Pham and Jha introduce MS-MAC, an S-MAC based protocol that adapts S-MAC's listening, sleeping and synchronization cycles to anticipated node movements. Node displacement is calculated from changes in signal strength; in case of rapid movements, a sending node hastens packet transmissions before a connection is lost.

In this paper, we build on this past body of work, utilizing the idea of adapting to current network conditions and specifically focusing on adapting *LPL* MAC protocols, which have proven quite energy-efficient for low data-rate wireless sensor network applications.

III. MiX-MAC: A Highly Adaptable MAC Protocol

A. Principles of MiX-MAC

No protocol in the *LPL* family outperforms the others over all potential conditions in the network. Selecting a MAC protocol supposes a compromise between excellent performance under certain circumstances (hoped to be the common case), and suboptimal operation otherwise. Various protocols may perform differently according to the broadcast / unicast nature of the exchanged packets, the size of the packets, or whether a node is mostly receiving or sending packets. Adapting the MAC schedule allows optimal performance across those parameters.

Additionally, adaptation must occur during runtime since the traffic patterns in the network may not be known *a-priori*. In a tracking application for instance, the appearance of an object modifies the ratio of broadcast-to-unicast packets and their sizes. A MAC protocol chosen for its good performance when no object is detected would probably be sub-optimal when a target is being tracked.

We propose creating a pool of MAC schedules that are *compatible* with one another: while the sender may decide which schedule to follow based on the parameters mentioned above, the receiver need not be informed of the changes in MAC schedules. For instance, a sender choosing a certain MAC schedule may expect an ACK frame between packet

transmissions; it will thus stay in receiving mode for a given time before it returns to transmitting mode. At the other end of the communication, a receiver simply wakes up periodically, and occasionally receives packets. If a received packet is marked with an acknowledgment request, it immediately sends an ACK frame. Switching between interchangeable MAC schedules guarantees that gains in energy and latency are achieved without any overhead other than the computation required to determine the best schedule to use. We call this approach, whereby the MAC schedule is adapted over time, MiX-MAC.

A small look up table within MiX-MAC helps in deciding what schedule is best suited for the current node, network and application conditions. This solution proves inexpensive in terms of computation power during runtime. The threshold values dictating a change in the MiX-MAC schedule can be established before deployment using simulation and implementation results. As we will show, inaccuracies in the estimates of current node, network and application conditions do not have a major impact on the performance of MiX-MAC.

Existing MAC protocols were included as part of the pool of compatible MAC schedules: X-MAC [8] and SpeckMAC [12], which were introduced around the same time. However, we also added an adaptation of CSMA-MPS, called MX-MAC.

B. X-MAC: A Short Preamble MAC Protocol

Under the X-MAC [8] schedule, a sender repeats the transmission of an advertisement packet containing the address of the intended receiver. Upon hearing the advertisement packet, the receiver replies with an ACK, which is followed by the transmission of the data packet by the sender. Figure 1 illustrates this process.

In [8], Buettner et al. do not propose implementing X-MAC for broadcast packets. X-MAC cannot broadcast packets as is, as the flow of advertisement packets cannot be answered by an ACK packet. A natural extension to X-MAC is to repeat advertisement packets for t_i and then send the data packet; however, receiving nodes have to wait until the completion of the advertisement cycle before they can receive the data packet—and go back to sleep (on average, they must wait for $\frac{t_i}{2} + t_{txPacket}$). In such cases, X-MAC performs equally to B-MAC, with the added advantage that it can be implemented using fixed preambles.

C. MX-MAC: a LPL Variant of CSMA-MPS Compatible with X-MAC and SpeckMAC Schedules

In [8], Buettner et al. make a convincing case for the energy and latency gains achieved by their proposed X-MAC protocol. Although efficient for unicast packets, this simple scheme is not well suited for broadcast transmissions. One additional drawback to X-MAC is its sensitivity to the hidden node problem and the persistence of a high risk of false positive packet reception acknowledgements. Indeed, early ACKs are sent and received before the data packet is transmitted, which does not guarantee successful reception of the packet.

Although it was introduced prior to X-MAC, CSMA-MPS [11] can be seen as a modification of X-MAC suitable for broadcast transmissions. CSMA-MPS repeats the data packet with its own wake-up schedule information and waits for ACK frames between transmissions. A received ACK signifies that the data packet has been correctly received and stops the transmission flow of data packets. This renders the MAC protocol immune to false positive packet receptions.

Although Mahlknecht et al. do not mention this point directly, the MAC schedule they propose can be adapted to broadcast packet transmissions so long as the sender does not request acknowledgment of the frames. Consequently, multiple receivers of the same packet may wake up, stay in RX mode until the full reception of a packet, and go back to sleep.

However, because CSMA-MPS must include scheduling information in every frame, which we do not need thanks to the implicit synchronization presented in Section V, and because it decouples channel probes from transmissions (much like WiseMAC, but unlike X-MAC), the MAC schedule presented by Mahlknecht et al. is not fully compatible with X-MAC and SpeckMAC. This is the reason why Langendöen [5] classifies it on the *PS* branch of MAC protocols. Therefore, we introduce MX-MAC, the *LPL* pendant to CSMA-MPS. Figure 1 illustrates the timeline for MX-MAC. In MX-MAC, the data packets contain no scheduling information, and a node may wake up only once per t_i period to probe the medium and possibly send a packet immediately following the probe.

D. SpeckMAC: Repeating the Data Packet

Another medium sensing protocol is SpeckMAC (precisely, SpeckMAC-D) [12]. In SpeckMAC, if a sender wants to transmit a packet to a receiver, it performs a clear channel assessment (CCA), and if successful, starts repeating the packet for at least t_i seconds. When a receiver wakes up, it checks the medium. If busy, it listens until it has received a full data packet or until it realizes that it is not the intended destination for the packet. Figure 1 illustrates the transmission schedule for SpeckMAC.

MiX-MAC alternatively uses the MAC schedules of X-MAC / C-MAC, MX-MAC or SpeckMAC based on a look-up table. In order to implement MiX-MAC, we must populate the table and find the appropriate switching thresholds through simulations and actual implementation of various scenarios, as discussed next.

IV. TinyOS Implementation of *LPL* Protocols

This section compares *LPL* MAC protocols implemented in TinyOS for the Tmote Sky platform in order to find the switching thresholds of the MiX-MAC look-up table.

A. Reconstruction Model

We accurately evaluate the lifetime of a mote by measuring the energy consumed under various basic operations using a fast data acquisition board. We measured the energy and time spent probing the medium, starting a transmission, sending one packet and switching the radio back to TX mode, stoping a transmission after a successful and failed (only for X-MAC and MX-MAC schedules) transmission, and receiving a packet.

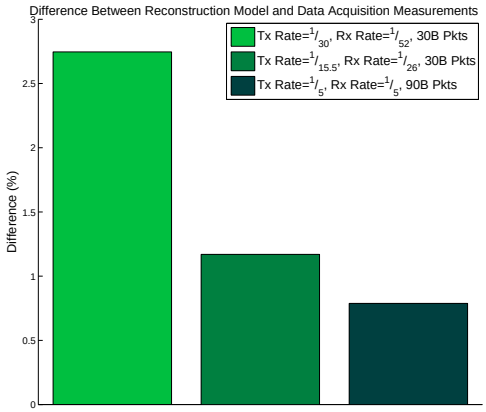
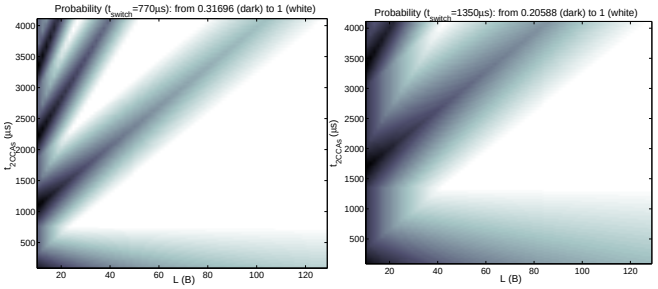


Fig. 2. Relative difference between real life scenarios and their prediction through the reconstruction model for various values of the rates of sent and received packets.



(a) (b)

Fig. 3. Probability to successfully hear an ongoing stream of packets as a function of the packet size (L) and t_{2CCAs}, for an RX / TX switch time t_{switch} = (a) 770 μs (SpeckMAC), and (b) $1,350$ μs (X-MAC / MX-MAC).

Every scenario is then reconstructed with Matlab by adding the energy expended during each operation.

Our measurements show that it takes a little under 32 μs to send one byte (we measured 31.89 μs exactly), which is confirmed by the CC2420 datasheet. Moreover, our measurements allow us to determine the *radio switch* time (after sending a frame, the CC2420 radio automatically switches to RX mode, therefore it must be turned back to TX mode before the next packet transmission) for each protocol. For SpeckMAC based schedules, this time is 772 μs, and it is 1.351 ms for MX-MAC and X-MAC—as they have to listen for ACK frames between packets. These radio switch times depend heavily on the TinyOS code and may differ from one programmer's implementation to the next.

In order to verify the accuracy of our reconstruction technique for determining node lifetime, we ran several scenarios where we directly measured node lifetime. Based on these experiments, the error of our reconstruction model does not exceed 3%, as shown in Figure 2. We have observed that most of the error emanates from the estimation of the idle power, which tends to vary over time due to temperature changes in the acquisition circuit.

B. Protocol Design Choices

We tried to optimize as many aspects of the MAC schedule as possible: the time separating two clear channel assessments (CCA) as well as the number of CCAs when sensing the medium, the number of CCAs before a packet transmission, the behavior of a node when it detects another ongoing transmission (what a sender should do when hearing another stream of packets during its switch to RX mode after every frame), etc. Since our goal is only to *compare* MAC protocols without bias toward one schedule, we endeavored to optimize the behavior of all three MAC protocols. Because all MAC schedules are meant to be compatible, they were implemented by the same TinyOS code. Consequently, all three protocols have the same essential parameters such as the number of CCAs and the time separation between them.

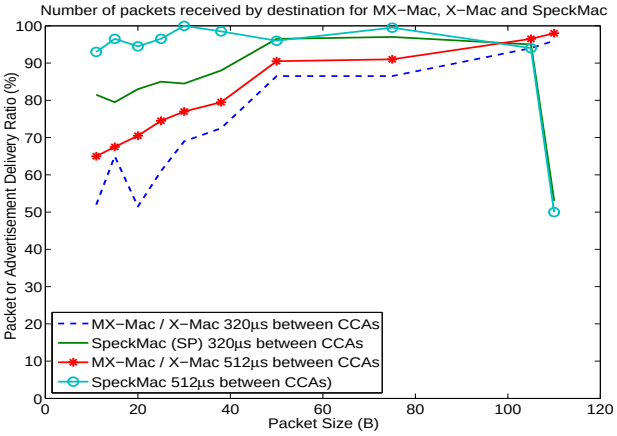
1) Time Separation Between CCAs: Two CCAs are usually sufficient to detect an ongoing transmission, provided they are separated by the correct time. We developed an analytical model to calculate the probability that a node would correctly hear an ongoing stream of packets as a function of the time between CCAs (t_{2CCAs}), the sender's *radio switch* time, and the size of the packets. Figures 3(a) and 3(b) plot the numerical value of this probability of successfully receiving the frame for the two radio switching times that we measured (770 μs and $1,350$ μs). These figures show that good choices for t_{2CCAs} tend to be around 600 μs and $1,100$ μs. From the perspective of energy consumption, a shorter time separation between consecutive CCAs is beneficial because it shortens the time spent probing the medium.

Our Tmote Sky implementation helped us refine these values: because this model is an ideal representation of the radio, we coded values for t_{2CCAs} that were much smaller than 600 μs and $1,100$ μs in order to account for the slower execution of the whole protocol stack on the Tmote Sky. Figure 4(a) shows the packet delivery ratios for two nodes randomly sending 100 packets to each other, including collisions, missed packets, bad radio states, etc. As the packet size increases, it is generally easier for a receiver to hear a transmission, which is confirmed by our analytical model. The dotted line shows a value for t_{2CCAs} that was not retained because of poor reliability.

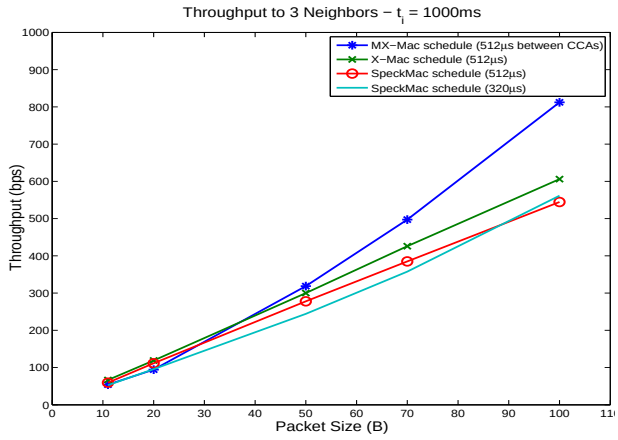
We found that an acceptable coded value for t_{2CCAs} is between 320 μs and 512 μs for SpeckMAC, and 512 μs for MX-MAC and X-MAC, which represent the best compromise between energy use in very low traffic networks and fairness to all protocols. For MiX-MAC, which must use compatible parameters for all MAC schedules, we set t_{2CCAs} to 512 μs for all protocols.

For packet sizes close to their maximum value (128 B), we found the radio to "jam" under SpeckMAC: the radio would issue RXFIFO overflows because the FIFO was filled before it could be read, and hence the packet delivery ratios in this case dropped significantly.

2) The X-MAC Design and Choice of Advertisement Packet Size: As Figure 4(a) shows, the packet delivery ratio for packets of size 11 B is only 65% for X-MAC, even with t_{2CCAs} set to 512 μs. This prompted the choice of larger advertisement packets (40 B), as is supposed by C-MAC [10]. In order to remain compatible with the other protocols, we considered all 40 B long packets to be advertisements. The



(a)



(b)

Fig. 4. (a) Comparison of the packet delivery ratio of the MAC schedules as a function of packet size and time between CCAs. (b) Throughput for $t_i = 1\ s$.

receiver, upon reading a $40\ B$ packet from its RXFIFO stays awake to receive the subsequent data packet. In other words, when a MAC protocol needs to send a $40\ B$ packet, it has to use the X-MAC schedule.

3) MAC Schedule Compatibility: Through design choices, we allowed the three MAC protocols to be compatible. The same TinyOS code can let a mote send and receive packets using the MX-MAC, X-MAC or SpeckMAC schedules.

More importantly, the basic principle behind schedule compatibility is that a receiver does not need to know the ongoing schedule, and simply ACKs packets that request it. For MX-MAC packets and for X-MAC advertisements, the *acknowledgement request* field must be set to one. If no ACK is requested, the receiver simply turns off after the packet has been received.

C. Determination of the Switching Thresholds

In order to populate the MiX-MAC look-up table, we must compare the X-MAC, MX-MAC and SpeckMAC-D schedules and determine which is most appropriate to the current set of parameters on the network.

1) Reliable Throughput or Goodput: In order to evaluate the throughput of the MAC protocols, we let one node send 100 packets to three different neighbors. Retransmissions may occur for MX-MAC and X-MAC, and $250\ ms$ after a successful transmission, a stream of packets is sent to the next neighbor, resulting in packet rates sometimes higher than $^1/_{t_i}$ for MX-MAC and X-MAC, as these protocols are interruptible and thus do not always transmit for the full t_i period. This should ensure that the application packet generating rate is not a limiting factor.

Figure 4(b) shows the number of bits transmitted in one second. For larger than $20\ B$ packets, MX-MAC performs best. This is because MX-MAC and X-MAC are interruptible, and thus take on average $^{t_i}/_2\ s$ to send a packet, and can attempt the next packet transmission $250\ ms$ later provided the source node is sending to different destinations. As the packet size increases, packets are received more reliably, which increases the goodput by up to 50%.

Since X-MAC uses fixed sized advertisement packets ($40\ B$), its throughput increases linearly with the data packet size as shown by the figure. However, its performance is not equal to that of MX-MAC since MX-MAC's larger packets get transmitted more reliably (as shown in Figure 4(a)). For a different reason, SpeckMAC based schedules are also linear: SpeckMAC can send only one packet per t_i period. When the packet size increases, so does the throughput.

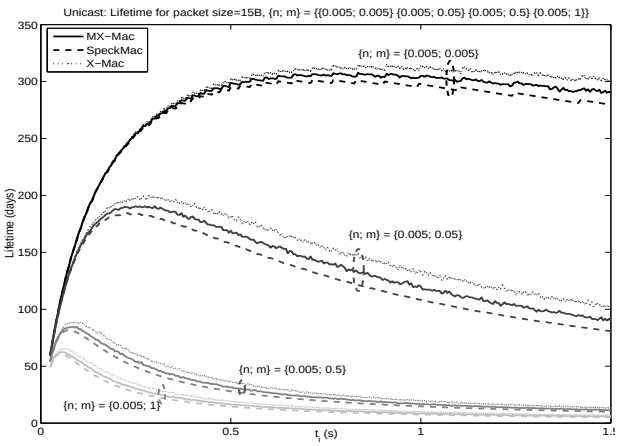
These results teach us that MiX-MAC can select the MAC schedule that will yield the best goodput for a certain packet size and t_i value. The results in Figure 4(b) suggest that for $t_i = 1\ s$, the X-MAC schedule yields the best throughput for small packets (less than $40\ B$), while the MX-MAC schedule has the best performance for larger packets.

2) Lifetime for Unicast Packets: Figure 5(a) shows the lifetimes for MX-MAC, X-MAC, and SpeckMAC schedules on the Tmote Sky platform (powered by two high capacity AA batteries of $4\ Ah$) for various $\{n; m\}$ values and $15\ B$ unicast packets—n designates the rate of received packets, and m[3] the rate of transmitted packets. SpeckMAC does not perform as well as MX-MAC and X-MAC. Figure 5(a) shows that X-MAC performs best for smaller packets. This holds only when the node is mostly sending. This is because the advertisement packet size is $40\ B$ (not the original X-MAC's $11\ B$), which increases the chance of being heard during a transmission, and thus saves retransmissions. At the same time, an increase in packet size increases the energy consumption by only $3-4\%$. This is because the radio transmits for $t_i\ s$ or until interrupted, whatever the packet size.

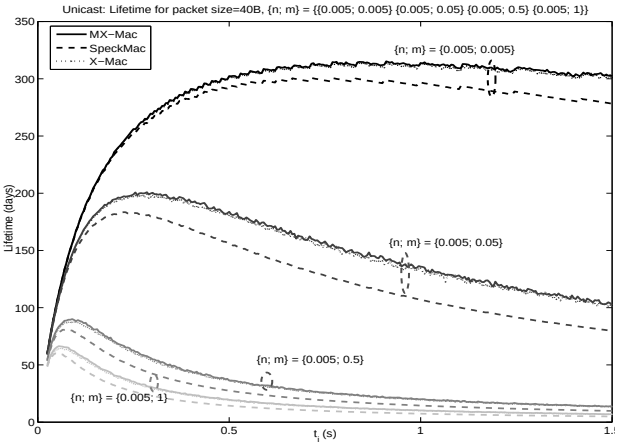
The advantages of X-MAC are reduced further when the data packet size reaches that of the advertisement size because the advertisement packet is no longer easier to hear than the data packet.

This section shows that for packets smaller than $40\ B$, and for cases when the node is mostly sending, X-MAC allows the node to increase its lifetime. In other cases, MX-MAC leads to a longer lifetime. If the main concern of the network

[3] m and n may be the same if the node is a relay that does not introduce new packets onto the network. $n = 0$ would typically designate a source node, and $m = 0$ a sink node.

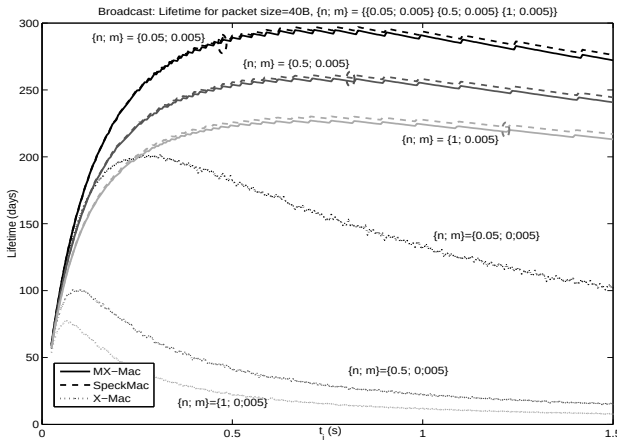


(a)

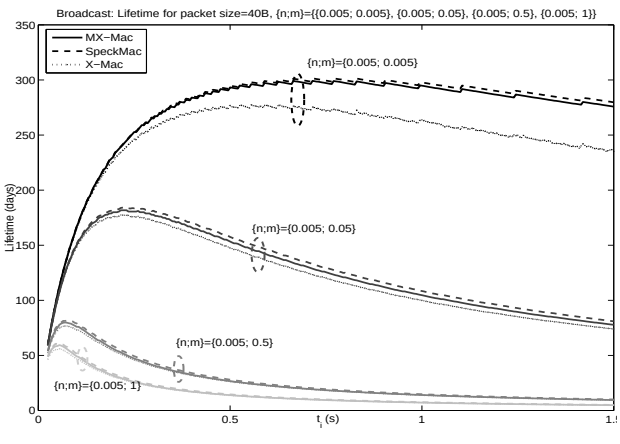


(b)

Fig. 5. Comparison of the MX-MAC, X-MAC and SpeckMAC schedules for unicast packets for scenarios where the node is mostly sending. Packets are $15\ B$ (a) or $40\ B$ (b).



(a)



(b)

Fig. 6. Comparison of the MX-MAC, X-MAC and SpeckMAC schedules for $40\ B$ broadcast packets when the node is (a) mostly receiving (b) mostly sending.

application is lifetime, MiX-MAC can thus switch between X-MAC and MX-MAC schedules under the conditions of Figures 5. This is possible because while the receiver does not get to pick the MAC schedule, the sender can select the appropriate MAC given current network and neighbor conditions. The receiver does not need to be informed of any changes in MAC scheduling. Based on the packets received, the receiver knows which schedule the transmitter is following.

3) Lifetime for Broadcast Packets: Contrary to unicast packets, one MAC schedule consistently spares the energy of the node, over the range of packet sizes. Figure 6(a) shows that X-MAC performs very poorly, as expected (Section III-B). Moreover, the lifetime increase provided by SpeckMAC is modest (2%) when the node is mostly sending as shown in Figure 6(b), and larger (10%) when the node is mostly receiving.

These relatively small lifetime increases hide the fact that with the SpeckMAC schedule, the destination nodes are much more likely to correctly receive the packets (Section IV-B1). Thus, using the SpeckMAC schedule for broadcast allows for longer lifetime and more reliable communication.

The results for the broadcast case show that MiX-MAC should always select the SpeckMAC schedule: as it will enjoy small gains in lifetime, and it will greatly improve packet delivery reliability compared to MX-MAC and X-MAC.

D. MiX-MAC Achieves the Upper Bound of Node Lifetime

In order to show the benefits of MAC schedule adaptation, we present the lifetime of a node sending unicast packets of different size. MiX-MAC selects either the MX-MAC or X-MAC schedules based on the packet size.

1) Picking the right MAC schedule: The previous results show that all MAC protocols sacrifice performance in unicast mode to that of the broadcast mode or vice-versa. MiX-MAC performs well against every combination of parameters because it constantly picks the best MAC schedule for these parameters.

MiX-MAC adopts SpeckMAC's schedule for broadcast packets, and for unicast packets, it uses four axes to decide the appropriate schedule. These include t_i value, packet size, estimated ratio of transmitted vs. received packets, and the ACK requirements determined by the upper level protocols or services.

The simplified look-up Table I gives the optimal schedule derived from reconstruction results as a function of several

TABLE I
LOOK UP TABLE TO DETERMINE WHICH PROTOCOL (SPECKMAC-D (S), X-MAC (X), OR MX-MAC (M)) PERFORMS BEST IN TERMS OF LIFETIME.

Pkt size	Unicast				Broadcast
	Mostly Sending $m = 0.005$ $n =$		Mostly Receiving $n = 0.005$ $m =$		
	0.05	1	0.05	1	
15 B	X	M	X	X	S
40 B	X	M	X	X	S
80 B	M	M	M	M	S
120 B	M	M	M	M	S

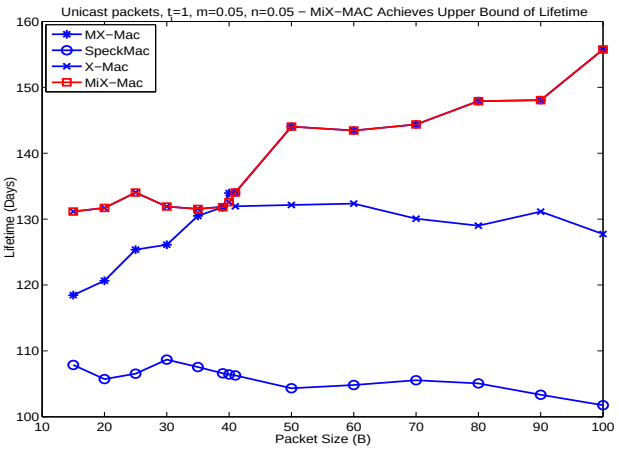


Fig. 7. A simple mapping function lets the protocol switch between schedules in order to increase the lifetime.

parameters. There exists an inherent trade-off between the size and complexity of the look-up table, and the granularity of switching MAC schedules over all the considered parameters.

2) Implementation Example: In this section, we present the node lifetime as a function of packet size in Figure 7 as an easy-to-read graph of schedule switching benefits.

The figure shows that MiX-MAC increases the lifetime by up to 30% compared to using the fixed schedules of X-MAC, MX-MAC, or SpeckMAC. While the mapping may not be perfect, the error of the point at which schedules are switched stems from the fact that schedules have very similar energy patterns at these packet sizes. The error in schedule switching is thus inherently small.

As can be seen in the figure, for the values of $\{m; n\}$ presented here, the lifetime may actually increase with the size of the packets sent. This is because a sending node is on for a fixed amount of time, and more reliable communications help avoid retransmissions as shown in Section IV-C1.

3) Effects of erroneous estimates resulting in suboptimal scheduling decisions: Since MiX-MAC picks schedules from a pool of existing protocols, erroneous estimates for the t_i value that yields the longest lifetime (due, for example, to an inaccurate measurement of m or n) would equally affect the performance of X-MAC, MX-MAC, and SpeckMAC.

However, if the lookup table were to point to an incorrect schedule, due for instance to an outdated or inaccurate estimate of the number of transmissions over receptions ratio, the network would simply operate at the level of performance of the chosen MAC protocol, without further degradation because of the absence of packet overhead for MiX-MAC. For small estimate errors around the points where schedules are switched (the intersections in Figure 7), the difference between the energy consumption of the optimal schedule and that of other schedules is small. A small error around these points cuts lifetime by only a few percentages, as our experiment suggests.

Elsewhere, a small estimate error has no effect: because the error is too small to impose a different schedule, the sending node picks the same schedule as the optimum. For very large estimate errors (for which the point of switching schedules is between the estimate and the actual value of a parameter), the resulting performance loss may be significant; however, large estimate errors (over 20%) should be rare by nature.

With this section, we saw that adapting the best-suited MAC schedule could increase node lifetime by up to 30%. The focus that we adopted was that of a node, without consideration for its neighbors. This meant that the transmit / receive schedules of nodes along a path were independent of one-another. We now shift focus to whole paths and try to synchronize nodes along routes in order to obtain further energy savings.

V. NODE SYNCHRONIZATION ALONG A PATH

One of the major drawbacks of *LPL* MAC protocols is that they tend to place a significant burden on the sending node for medium to low duty cycles, even if we adapt the transmission schedule via MiX-MAC. The choice of a network-wide t_i value is a delicate decision: a programmer may wish to elect a large value, but such a low duty cycle may waste energy when transmitting packets, making it an unlikely choice. While researching MiX-MAC, we conjectured that certain *LPL* protocols could be synchronized in a way that would allow staggering wake-up schedules. This section details the various synchronization techniques that can be used to minimize delays and energy consumption. While synchronization along a path comes with virtually no overhead, the rare case when packets need to be exchanged on a bidirectional path requires a small amount of overhead (three extra packets per bidirectional path).

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 MiX-MAC with only the MX-MAC schedule, although the results in this section of the paper can be easily extended to the whole family of *int-LPL* protocol schedules. 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, Section IV showed that the small advertisement packets in X-MAC can be hard to hear, 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

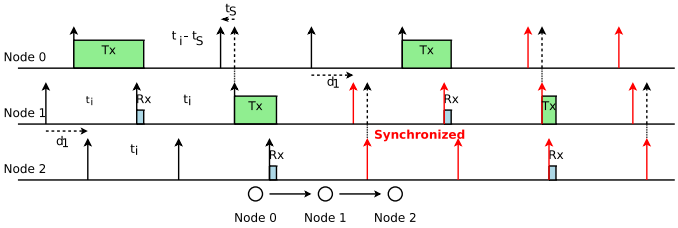


Fig. 8. Synchronization principle for three nodes running an *int-LPL* protocol.

with X-MAC. We thus selected MX-MAC with 50 B packets, although the principle of transmission / reception schedule adaptation holds for all *int-LPL* protocols. 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 92% over a four-hop path, assuming independent links. For simplicity purposes, packets are delivered in a best-effort manner, and unsuccessful transmissions result in dropping the packet.

A. Synchronization Over a Unidirectional Path

This section expands our previous work on path synchronization [17] in order to study how to fuse the benefits of such an approach with MiX-MAC.

1) Principle: Under the MX-MAC schedule, a node learns 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 transmitting the ACK frame following the reception of a packet, node 1 sets its timer to wake up t_i s later. The sending node 0 also sets a timer (upon receiving the ACK), in this case for t_i minus 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 (one frame) and send an ACK frame. This allows node 0 to wake up slightly before node 1 during the next rounds, thus reducing the time for which node 0 is transmitting.

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 sink. 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 than two 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 (*i.e.*, received an ACK). For the case of three nodes, full route synchronization is not achieved until after two packets have been sent, as illustrated in Figure 8.

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 simulation results, can be modeled as follows.

Let $n = h$ be the number of hops from node 0 to node n. τ_k^j designates the time, modulo t_i, at which node k wakes up to probe the medium or send a packet, and after it has sent the j^{th} packet. At the beginning, wake-up times are separated by random periods of time. The effects of missing the beginning of a transmission (causing the receiver to receive the next frame in the stream) are negligible compared to t_S. 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. The propagation of the first packet over the path leads to changes in the nodes' wake-up times as follows:

$$\begin{aligned}\tau_0^1 &= \tau_1^0 - t_S \\ \tau_{k-1}^1 &= \tau_k^0 - t_S \\ \tau_{n-1}^1 &= \tau_n^0 - t_S = \tau_n - t_S\end{aligned}$$

After the second packet is sent, the last three nodes are synchronized (τ_n, τ_{n-1} and τ_{n-2}):

$$\begin{aligned}\tau_0^2 &= \tau_1^1 - t_S \\ \tau_{n-2}^2 &= \tau_{n-1}^1 - t_S \\ &= (\tau_n - t_S) - t_S = \tau_n - 2t_S\end{aligned}$$

Therefore, after the j^{th} packet, the last $j+1$ nodes are synchronized, and we have $\tau_{n-l}^j = \tau_n - lt_S$ for all $l \leq j$. Other nodes $k = n-l$, such that $j < l \leq n$, still have random time separations between their wake-up times. The nodes are all synchronized when $\tau_0^n = \tau_n - nt_S$ after at most $j = n$ packets have been properly sent[4].

Once the path is synchronized, the end-to-end delay can be expected to be equal to $t_S + (n-1)(t_i + t_S) + t_{Rx}$, as suggested by Figure 9.

This short analysis also shows that clock drift has little 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. The measured clock drift for the Tmote Sky is at most 5 ppm. This means that the relative drift between two motes is $\alpha_{drift} \leq 10^{-5}$. If $\mathcal{T}$ is the time between two packet streams, then we must have $\mathcal{T} < \frac{(t_S - t_{Rx})}{\alpha_{drift}}$. In our implementations, we commonly used a t_S value of 50 ms, leading to $\mathcal{T} < 3,600$ s: in order to guarantee the proper preservation of the nodes' synchronized wake-up schedules, a unicast packet must be sent at least every hour on the path. For WiseMAC, $\mathcal{T}$ is only half that because nodes use absolute schedules to synchronize themselves and because they must avoid overhearing. Therefore, nodes running WiseMAC should start sending $2t_S$ before their next-hop wakes up. The results in Section VI-A3 take this into consideration.

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

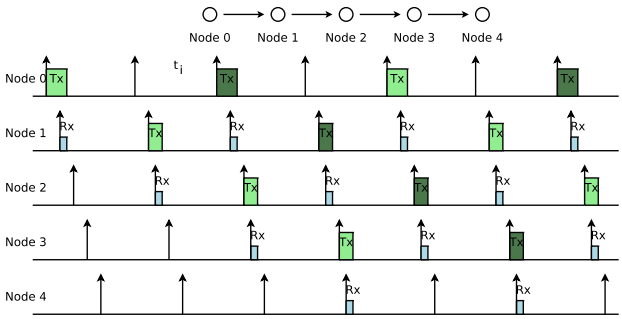


Fig. 9. Node 0 pipelines packets and increases the packet rate.

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 routing protocol to request relaying the packet. When the "send" command is issued, the MAC protocol immediately starts the stream of packets. In order to be successful, the packet transmission must start before the next hop probes the channel. 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 $nt_S + t_{Rx}$.

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 $t_{Rx} + t_p < t_S$.

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.

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 9. Pipelining is only possible with synchronized nodes because if nodes are not synchronized and try to transmit packets every $2t_i$ s, they would interfere with one another and exacerbate the hidden node problem, common to all *LPL* protocols.

B. 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 (n_f) parallel branches: nodes i hops away from the destination tend to wake up at the same time, causing contention. This node configuration is illustrated by Figure 10(a).

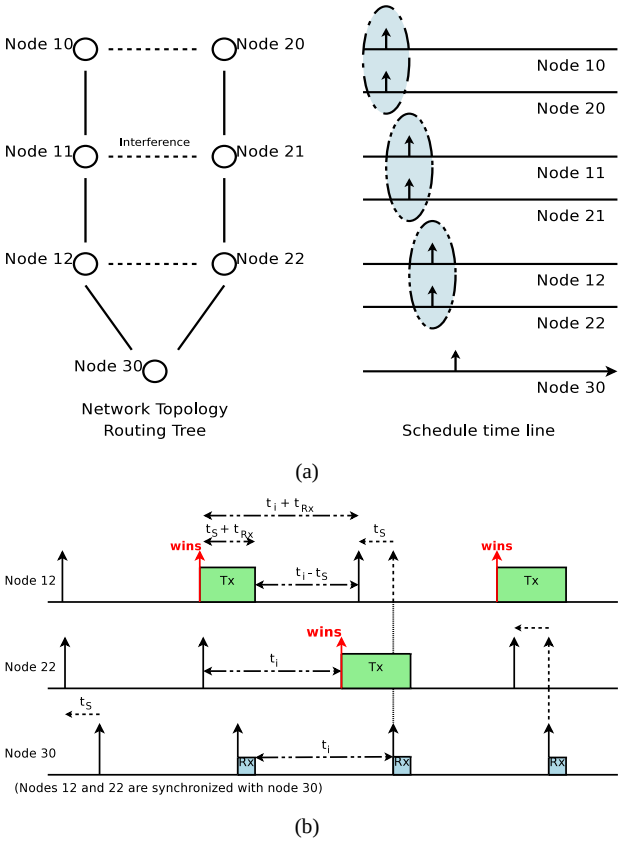


Fig. 10. (a) Synchronized nodes along two parallel paths: nodes $\{10, 11, 12, 30\}$ form one path, and $\{20, 21, 22, 30\}$ another one. The dotted lines indicate that the nodes can communicate with each other (and thus interfere). (b) Mitigation of the problem.

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 or aggregation strategy is employed) and the application (which may require high data rates from co-located 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 as Nodes 12 and 22 are in Figure 10(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. In effect, after successfully transmitting a packet, a node's wake-up schedule gets delayed by t_{Rx}, which separates it from other contenders and allows other flows access to the common destination. This process is not dependent on the number of flows converging to the same node, rather, t_{Rx} limits flow fairness as it should be above $\sim t_i \cdot n_f \cdot 2\alpha_{drift}$. Figure 10(b) illustrates this with a time line: after sending a packet, nodes 12 and 22 are separated by

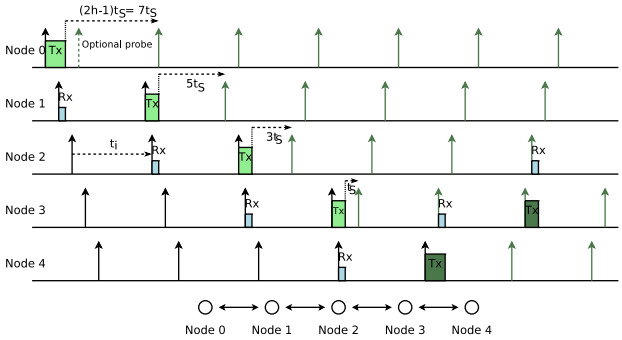


Fig. 11. Bidirectional path synchronization time line.

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.

C. 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 similar to the crossed-ladders pattern of [13] and is available (with simulation results) in [17]. We provide Figure 11 to illustrate this bidirectional synchronization process.

VI. Simulation and Implementation of Synchronization Principles

A. Simulations

In this section, we explore the advantages and limits of node synchronization through Matlab simulations. We use the same accurate Matlab model for time and energy consumption as that of Section IV. 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 40 B packets at a rate of $^1/_2$ $ptk.s^{-1}$. The time separating two frames in the same stream is 1.351 ms, and the synchronization back-off t_S is 50 ms.

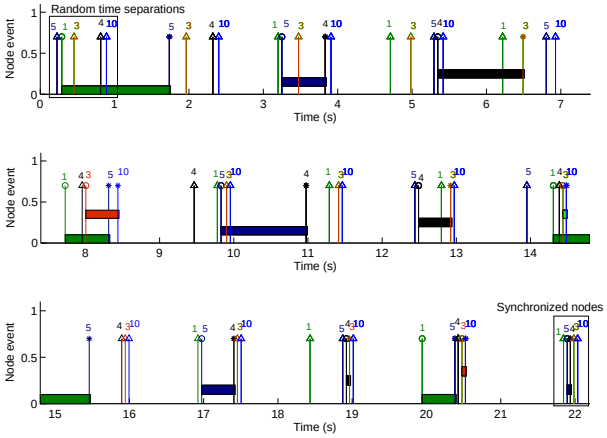


Fig. 12. On the path {1, 5, 4, 3, 10}, the nodes synchronize correctly after only a few packets.

1) Synchronization Principle: We simulated a scenario in which $t_i = 1.5$ s. Notice that the t_S value is about twice the reception time of a 50 B 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 12 shows that five 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.

2) Packet Delay: Next, we investigate the packet delay after the nodes have been 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 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 [1].

a) Delay of Non-Urgent Packets: Figure 13 shows the packet delay of non-urgent and urgent packets. The first eight packets show how wake-up schedule synchronization reduces the delivery delay for the node configuration of the previous section. 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, to account for randomly waking-up during frames).

The first packet is sent without any synchronization between the nodes, and its delay is 9.7 s, a value that depends on the initial random wake-up times. In this case, synchronization cut

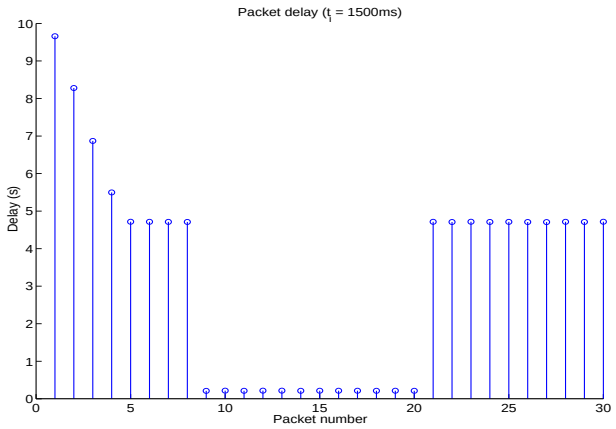


Fig. 13. Packet delay on the same path as Figure 12. Packets 9 through 20 are marked as urgent.

TABLE II
ENERGY CONSUMPTION AND PACKET DELAY.

		MX-MAC				WiseMAC	
		Sync		Non-sync			
Parameter		$\overline{Energy}$ (J)	$\overline{Delay}$ (s)	$\overline{Energy}$ (J)	$\overline{Delay}$ (s)	$\overline{Energy}$ (J)	$\overline{Delay}$ (s)
h	1	1.59	0.065	12.73	0.76	2.53	0.13
	2	1.50	1.63	9.76	2.91	2.57	0.87
($t_i =$	3	1.49	3.21	9.37	6.10	2.60	1.52
1.5 s)	4	1.54	4.91	9.00	10.63	2.61	2.28
	5	1.45	6.34	8.80	13.54	2.61	5.15
t_i	0.5	1.77	1.74	5.73	3.08	2.72	1.12
	1	1.67	3.26	8.99	6.62	2.62	2.11
($h = 4$)	1.5	1.54	4.91	9.00	10.63	2.61	2.28
	2	1.31	6.26	9.10	13.98	2.59	2.92

the packet delay by over 50%.

b) Packet Delay of Urgent Packets: Packets 9 through 20 of Figure 13 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.

3) 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 II gives the average energy consumption of nodes on a path for MX-MAC and WiseMAC. 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 six 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 $t_i/2(t_S+t_{Rx})$.

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. The same is true of WiseMAC, which reduces energy consumption for lower duty cycles.

Compared to WiseMAC, MX-MAC with synchronization consumes 30% less energy because it combines probes and transmissions, and because it uses hardware acknowledgements, which allow shorter packet reception times. However, the delivery delay of regular packets of MX-MAC is larger than for WiseMAC (it would be smaller for urgent packets).

B. Implementation on Tmote Sky

We implemented the principles behind node synchronization in TinyOS for the Tmote Sky platform. We present results from this implementation.

1) Methodology: Once the MX-MAC code was set on the motes, 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, artificially degrading the performance of the protocol. However, because of the nature of synchronized 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 using the motes only. The Tmote Sky can only measure its internal voltage through the ADC, which is typically noisy, and the battery voltage does not evolve as a linear function of the remaining energy. Because the MX-MAC protocol is very energy efficient, the voltage drop over a

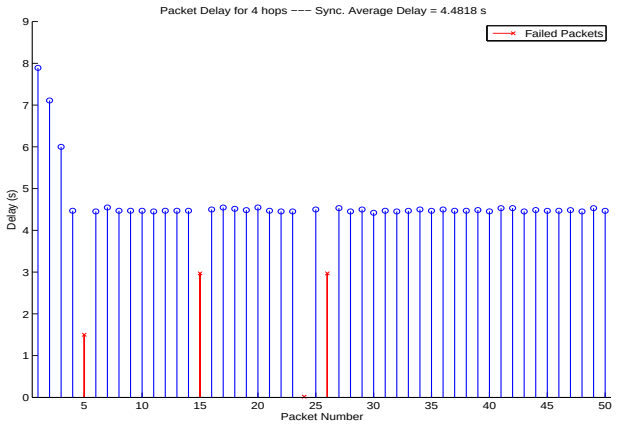


Fig. 14. Packet delay over the same path of the implementation (for failed packets, the delay corresponds to the delivery delay up to the last successful transmission on the path; failed packets do not reach the eventual destination and are not counted in the average delay).

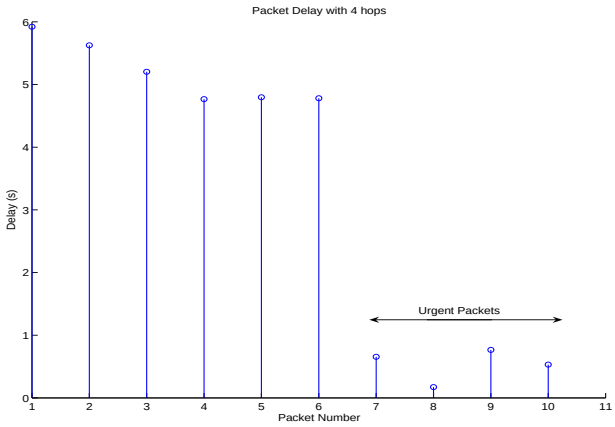


Fig. 15. Delay of non-urgent and urgent packets over a synchronized path.

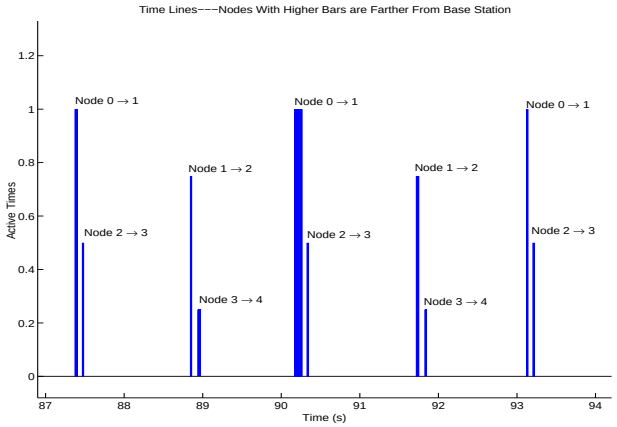


Fig. 16. Packet pipelining: node 0 sends a new packet every $2t_i$. Transmissions end up being staggered.

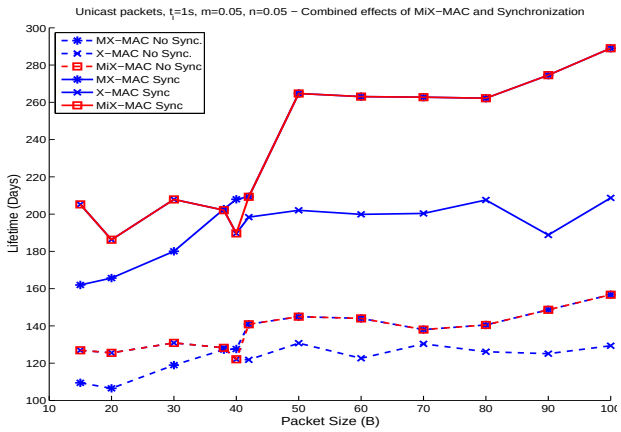


Fig. 17. Comparison of the lifetime of nodes running MiX-MAC with and without node synchronization.

practical period of time is well within the natural ADC noise, with or without node synchronization. This method would therefore be less precise than the reconstruction technique used throughout the paper and introduced in Section IV-A.

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

Figure 14 shows that the motes on the four-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).

3) Urgent Packets: Next, we present the delay of urgent packets in Figure 15 and confirm the results obtained through the reconstruction model. Packets 7 through 10 are marked as urgent, and their delay hovers between $766\ ms$ and $172\ ms$, the actual value of the delay being hard to measure due to the typically slow link between the mote and the PC. It also shows that urgent packets do not break the path synchronization.

4) Packet Pipelining: Finally, Figure 16 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 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.

These results show that node synchronization over a temporarily fixed path is practical.

C. Combined Effects of MiX-MAC and Node Synchronization

Figure 17 shows the cumulative increases in node lifetime with MiX-MAC and synchronization. The conditions of the simulation are the same as in Section IV-D2, although the results were averaged over fewer iterations. As in Figure 7, MiX-MAC achieves the highest lifetime for most packet sizes, with and without node synchronization, although synchronization greatly increases the lifetime of nodes running MiX-MAC (by up to 95%). When packets fail to be heard at the receiver, synchronization does not help in any way: the packet will have to be retransmitted, regardless of when the receiver woke up—and missed— the packet. However, since MiX-MAC selects the protocol with the highest delivery reliability for each packet size, packet failures are less common, and synchronization can then play its full role of saving energy and reducing packet delay.

D. Discussion

The TinyOS implementation demonstrated the feasibility of compatible MAC schedules and the benefit of switching between them. No extra overhead was required, save the memory required for the look-up table. MiX-MAC may adopt the MAC schedule most adapted to conditions in the network to increase lifetime or throughput.

On top of the gains obtained through MAC schedule adaptation, we also propose a simple approach to synchronizing nodes on a temporarily-fixed path for the sub-family of *int-LPL* protocols. 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 since it needs to be constant for only h packets.

Synchronization of transmit / receive schedules 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. Pipelining packets over synchronized paths doubles the packet rate. Urgent packets are delivered almost immediately, taking the delay from $t_S + (n-1)(t_i + t_S) + t_{Rx}$ to $nt_S + t_{Rx}$.

Although MiX-MAC and node synchronization may be implemented without the benefit of the other, their combined impact on node lifetime and packet delivery delay exceeds that of each approach independently. This is because MiX-MAC may select the most reliable MAC schedule, which in turns greatly facilitates node synchronization.

VII. Conclusions and Future Work

Existing MAC protocols employ identical schedules for both unicast and broadcast packet transmissions or, when impossible, simply modify their "unicast schedule" to work with broadcast packets. For instance, IEEE 802.11 cannot perform an RTS / CTS handshake for broadcast packets, and thus only utilizes CSMA for broadcast packets, regardless of the impact on lifetime or contention.

In this paper, we propose adapting the MAC schedule to node and network conditions to improve performance under a wide range of conditions and for both unicast and broadcast packets. The MAC schedule should be chosen to maximize the lifetime of the network, which includes reducing contention.

Through simulation and implementation on the Tmote Sky platform with TinyOS, we showed that MiX-MAC can significantly increase per-node lifetime and 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 increase the number of parameters and metrics to switch MAC schedules. We also plan to investigate further node synchronization for 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.

References

[1] C. J. Merlin and W. B. Heinzelman, "Duty cycle control for low power listening MAC protocols," in *Proceedings 5^{th} IEEE Conference on Mobile Ad-hoc and Sensor Systems (MASS'08)*, Sep. 2008.

[2] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of the 2^{nd} ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Nov. 2004, pp. 95–107.

[3] Y. Wei, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *Proceedings of the Twenty-First International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, Jun. 2002.

[4] T. van Dam and K. Langendöen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," in *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, Oct. 2003.

[5] K. Langendöen, "Medium access control in wireless sensor networks," in *Medium Access Control in Wireless Networks, Volume II: Practice and Standards*. Nova Science Publishers, 2007.

[6] A. El-Hoiydi and J. Decotignie, "WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks," in *Proceedings of the First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, Jul. 2004.

[7] X. Shi and G. Stromberg, "SyncWUF: An ultra low-power MAC protocol for wireless sensor networks," in *IEEE Transactions on Mobile Computing*, vol. 6, no. 1, Jan. 2007, pp. 115 – 125.

[8] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," in *Proceedings of the 4^{th} Embedded Networked Sensor Systems (SenSys'06)*, Nov. 2006, pp. 307–320.

[9] A. El-Hoiydi, "Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks," in *Proceedings of the IEEE International Conference on Communications (ICC'02)*, Apr. 2002.

[10] S. Liu, K.-W. Fan, and P. Sinha, "CMAC: An energy efficient MAC layer protocol using convergent packet forwarding for wireless sensor networks," in *Proceedings of the Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'07)*, Jun. 2007.

[11] S. Mahlknecht and M. Böck, "CSMA-MPS: A minimum preamble sampling MAC protocol for low power wireless sensor networks," in *Proceedings of the IEEE International Workshop on Factory Communication Systems*, Sep. 2004.

[12] K.-J. Wong and D. Arvind, "SpeckMAC: Low-power decentralised MAC protocol low data rate transmissions in Specknets," in *Proceedings 2^{nd} IEEE International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality (REALMAN'06)*, May 2006.

[13] A. Keshavarzian, H. Lee, and L. Venkatraman, "Wakeup scheduling in wireless sensor networks," in *Proceedings of the Seventh ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'06)*, May 2006.

[14] G. Lu, B. Krishnamachari, and C. Raghavendra, "An adaptive energy-efficient and low-latency MAC for data gathering in sensor networks," in *Proceedings of the Fourth International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN)*, 2004.

[15] T. Watteyne, A. Bachir, M. Dohler, D. Barthel, and I. Augé-Blum, "1-hopMAC: An energy-efficient MAC protocol for avoiding 1-hop neighborhood knowledge," in *Proceedings of the IEEE International Workshop on Wireless Ad-hoc and Sensor Networks (IWWAN'06) at SECON*, vol. 2, Sep. 2006, pp. 639 – 644.

[16] H. Pham and S. Jha, "An adaptive mobility-aware MAC protocol for sensor networks (MS-MAC)," in *Proceedings 1^{st} Conference on Mobile Ad-hoc and Sensor Systems (MASS'04)*, Oct. 2004.

[17] C. J. Merlin and W. B. Heinzelman, "Node synchronization for minimizing delay and energy consumption in low-power-listening MAC protocols," in *Proceedings 5^{th} IEEE Conference on Mobile Ad-hoc and Sensor Systems (MASS'08)*, Sep. 2008.