Ultra-Low Duty Cycle MAC with Scheduled Channel Polling*

Wei Ye, Fabio Silva, and John Heidemann
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292
{weiye, fabio, johnh}@isi.edu

Abstract

Energy is a critical resource in sensor networks. MAC protocols such as S-MAC and T-MAC coordinate sleep schedules to reduce energy consumption. Recently, low-power listening (LPL) approaches such as WiseMAC and B-MAC exploit very brief polling of channel activity combined with long preambles before each transmission, saving energy particularly during low network utilization. Synchronization cost, either explicitly in scheduling, or implicitly in long preambles, limits all these protocols to duty cycles of 1–2%. We demonstrate that *ultra-low* duty cycles of 0.1% and below are possible with a new MAC protocol called scheduled channel polling (SCP). This work prompts three new contributions: First, we establish optimal configurations for both LPL and SCP under fixed conditions, developing a *lower bound of energy consumption*. Under these conditions, SCP can extend lifetime of a network by a factor of 3–6 times over LPL. Second, SCP is designed to *adapt well to variable traffic*. LPL is optimized for known, periodic traffic, and long preambles become very costly when traffic varies. In one experiment, SCP reduces energy consumption by a factor of 10 under bursty traffic. We also show how SCP adapts to heavy traffic and streams data in multi-hop networks, reducing latency by 85% and energy by 95% at 9 hops. Finally, we show that SCP can *operate effectively on recent hardware* such as 802.15.4 radios. In fact, power consumption of SCP decreases with faster radios, but that of LPL increases.

Categories and Subject Descriptors: C.2.2 [Computer-Communication Networks]: Network Protocols; C.4 [Performance of Systems]: Performance Attributes

General Terms: Design, Algorithms, Performance, Experimentation, Measurement

Keywords: wireless sensor networks, medium access control, energy efficiency, ultra-low duty cycle, scheduling

*This research is partially supported by the National Science Foundation (NSF) under the grant NeTS-NOSS-0435517 as the SNUSE project, by a hardware donation from Intel Co., and by Chevron Co. through the USC Center for Interactive Smart Oilfield Technologies (CiSoft).

SenSys'06, November 1–3, 2006, Boulder, Colorado, USA.

1 Introduction

Energy is a critical resource in battery-powered sensor networks. Current applications such as habitat monitoring target sensor deployments of months or years [1, 15]. With small sensor nodes like Berkeley Motes the radio is a major source of energy consumption [7, 11]. On Mica2 motes the radio draws 22mW when idle or receiving and more when transmitting [9], which is similar to the CPU power consumption and much larger than other components. On MicaZ motes, the IEEE 802.15.4 radio draws even more power when active (56mW in receiving). Thus it is not surprising that developing protocols to optimize radio energy consumption has been a major research topic.

Major sources of energy waste are idle listening, collision, overhearing and control overhead [26]. Among them, idle listening is a dominant factor in most sensor network applications. The central approach to reducing energy lost to idle listening is to lower the radio duty cycle by turning the radio off part of the time. More formally, we define duty cycle as the ratio between listen time and a full listen/sleep interval. To keep the abstraction of a fully connected network[1], networks use duty cycling. Three approaches are generally used: TDMA, scheduled contention, or low-power listening. We do not consider TDMA here because of difficulties that arise in networks of peers (as in sensor networks) that lack centralized or cluster-based control, and at very low duty cycles.

One important approach is to *schedule* coordinated transmission and listen periods, as seen in S-MAC [26, 27], T-MAC [24], and TRAMA [19]. The schedule determines when a node should listen and when it should sleep. In S-MAC and T-MAC nodes adopt common schedules (possibly with help in [13]), synchronizing with periodic control messages (SYNC packets). A receiver only listens to brief contention periods, while senders contend during these periods. Only nodes participating in data transfer remain awake after contention periods, while others can then sleep.

[1] A complementary alternative is to break the abstraction of connectivity, leading to topology control protocols, outside the scope of this paper.

Scheduling reduces energy cost by ensuring that listeners and transmitters have a regular, short period in which to rendezvous and can sleep at other times. Overhead is due to schedule maintenance and listening during contention intervals if there is nothing to send. For networks with low utilization the sleep interval can be extended, but potentially reducing throughput when busy. (Optimizations such as FRTS [24] and adaptive listening [27] reduce this cost.)

Another technique is *low-power listening* (LPL), presented in WiseMAC [3] and B-MAC [17]. In LPL, nodes wake up very briefly to check channel activity without actually receiving data. We call this action *channel polling*[2]. If the channel is idle, the node immediately goes back to sleep. Otherwise it stays awake to receive data. Although nodes regularly poll the channel with a pre-defined polling period, their polling times are not explicitly synchronized. To rendezvous with receivers, senders send a long *preamble* before each message (longer than the polling period), which is guaranteed to intersect with a polling.

Since channel polling is about 10 times less expensive than listening for full contention period, LPL protocols consume much less energy than existing scheduled protocols in lightly loaded networks. Unfortunately, current LPL-based protocols have three major problems. First, receiver and polling efficiency is gained at the much greater cost of senders. In fact, the duty cycle is limited to 1–2% because the polling frequency needs to balance the cost on sending preambles and polling the channel. (Both WiseMAC and B-MAC can avoid long preambles in certain situations [3, 17, 18, 4], but not generally. We discuss details in Section 6.) Second, this balance between sender and receiver costs makes LPL-based protocols very sensitive to tuning for an expected neighborhood size and traffic rate. When the actual neighborhood or traffic does not match the ideal model, its performance is significantly reduced, particularly when traffic rates vary greatly. Finally, it is challenging to adapt LPL directly to newer radios like 802.15.4, since the specification limits the preamble size.

This paper introduces a new MAC protocol based on scheduled channel polling (SCP). By synchronizing the channel polling times of all neighbors, SCP-MAC eliminates long preambles in LPL for all transmissions, and is able to operate at ultra-low duty cycles when traffic is light. The key contributions of this paper are to address each of these challenges: understanding the optimal behavior of both scheduling and channel polling separately and together, to place a lower bound on energy costs; developing a protocol that adapts to dynamically changing traffic patterns efficiently; and understanding how these techniques apply both to existing (CC1000) and new (802.15.4, CC2420) generations of radios for sensor networks.

First, we discover *optimal configurations* for both LPL and SCP and their *lower bounds of energy consumption* with periodic traffic. These configurations allow us to answer the fundamental question about the relative benefits of scheduled access compared to asynchronous polling. While prior work showed optimal lifetime for LPL as a function of probe time and neighborhood size [17], we extend that work to provide a closed-form solution for optimal configurations (Section 3.2). More importantly, we present optimal configurations for synchronized channel polling to achieve its best performance (Section 3.3). We validate this analysis experimentally (Section 5.1). We show that optimal SCP can typically extend lifetime of a network by a factor of 3–6 times compared to optimal LPL. The end result of this analysis is to demonstrate that SCP-MAC can achieve *ultra-low duty cycles* of 0.1% or less, an order of magnitude or more better than current approaches.

Second, we design SCP-MAC to *adapt well to variable traffic*. While some applications, such as habitat monitoring [1, 15] have periodic traffic, many applications have varying or unpredictable traffic rates. For example, in event-triggered sensor detection, the network is quiescent for long periods when nothing is detected, then quite busy after detection, during tracking. LPL is optimized for known, periodic traffic, and long preambles become very costly when traffic varies. We demonstrate the energy benefits of SCP over LPL for both broadcast and unicast traffic under bursty rates. SCP is able to reduces energy consumption by a factor of 10 for broadcast traffic (Section 5.2) and by a factor of 20 for unicast with overhearing avoidance (Section 5.3). To cope with latency in multi-hop networks, we develop *adaptive channel polling* that allows the network to stream data under heavy traffic by dynamically adding high-frequency polling on nodes (Section 2.2). This approach reduces latency by 85% in a 9-hop network (Section 5.2).

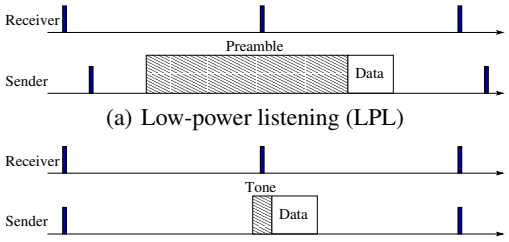
Finally, we show that SCP can *operate effectively on new radios* such as those supporting IEEE 802.15.4. Such radios have been perceived to work poorly with LPL because of limitations on control of preamble length. Although our implementation is very preliminary, we present early results showing that energy conservation is feasible with 802.15.4 radios, even without base-stations or asymmetric operation. Our analysis and initial experiments show that power consumption of SCP decreases with faster radios, while that of LPL-based approaches increases.

2 Design of SCP-MAC

SCP-MAC is designed with two main goals: first, to push the duty cycle an order of magnitude lower than is practical with current MAC protocols, and second, to adapt to variable traffic loads common in many sensor network applications.

We adopt several approaches to meet these goals. We begin by combining scheduling with channel polling in an optimal way that the energy cost can be minimized with periodic traffic. We derive and employ optimal intervals for schedule synchronization based on worst-case clock drift. This approach gains the benefits of brief channel polling, and replaces the penalty of long LPL preambles with a much smaller cost of synchronizing schedules. Finally, we develop a new algorithm to dynamically adjust duty cycles in the face of busy networks and streaming traffic, reducing the latency in multi-hop networks.

[2]The term "polling" is also sometimes used to indicate a base-station interrogating each client to see if it has data to send. All nodes in our system are peers of each other; in this paper polling refers only to each node sampling the channel to check for activity.



(a) Low-power listening (LPL)

(b) Synchronized channel polling (SCP)

Figure 1. Sender and receiver synchronization schemes.

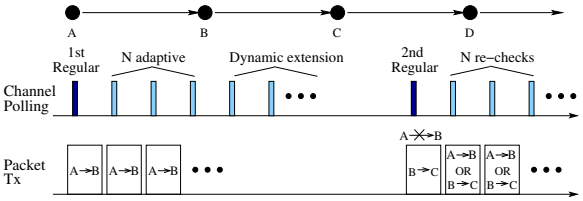


Figure 2. Adaptive channel polling and multi-hop streaming.

2.1 Synchronized Channel Polling

Channel polling reduces the cost of discovering traffic, since checking for the presence or absence of network activity is much cheaper than knowing what the activity is. In low-power listening (LPL), nodes poll channel asynchronously to test for possible traffic. To send a packet, the sender adds a preamble before the packet. This preamble is effectively a *wake-up signal*, informing other nodes a data packet is about to be transmitted. The preamble must be at least as long as the channel polling period to ensure all receivers will detect it (see Figure 1(a)). The performance of LPL is sensitive to the channel polling period, since longer periods reduce receiver costs but increase sender costs. Selecting an optimal value requires knowledge of network size and completely periodic traffic [17].

SCP-MAC adopts channel polling from LPL approaches. However, unlike LPL, SCP-MAC *synchronizes* the polling times (schedules) of all neighboring nodes. The primary advantage of scheduled polling is that only a short wake-up tone is required for senders to guarantee rendezvous. As an example, compare the wakeup tone duration in SCP (Figure 1(b)) with LPL (Figure 1(a)). In addition, synchronization reduces the cost of overhearing, since on average all nodes will hear half the preamble before waking up, even for packets addressed to other receivers. Moreover, with synchronization SCP works efficiently for both unicast and broadcast traffic, while some existing optimizations to improve LPL work only for unicast. Finally, as we will show later experimentally (Section 5.2), short wakeup tones make SCP-MAC more robust to varying traffic load.

The penalty of scheduled polling is the cost of maintaining schedule synchronization, and potentially the requirement of maintaining multiple schedules. SCP-MAC distributes schedules much as developed by S-MAC [26]: each node broadcasts its schedule in a SYNC packet to its neighbors every *synchronization period*. One of the major contributions of this paper is to discover the optimal synchronization period and wakeup tone length that minimize the overall energy consumption. The details of this analysis in Section 3 show that this control overhead is negligible, typically one packet every 10–60 minutes. In addition, SCP will *piggyback* schedule information on any data packets that happen to be sent. When the data rate is higher than the synchronization period, piggybacking can completely suppress explicit SYNC packets. When there is no opportunity to piggyback, a periodic timer forces transmission of SYNC packets as described in Section 4.3. Finally, we also expect to use prior work to allow all nodes to adopt a single schedule [13], thus avoiding unnecessary additional synchronization points.

2.2 Adaptive Channel Polling and Multi-hop Streaming

While traffic in some classes of sensor network applications is completely periodic, a much larger set of applications mix periodic and bursty traffic or consist of unpredictable traffic mixes. Object tracking is a worst-case application, since there is no traffic to send most of the time, but bursts of activity when a target is detected. Such a network does not have a single good operating point, since it must run at a low duty cycle to match long idle periods, but then is penalized with long preambles and expensive transmission costs during busy cycles. Furthermore, collisions are frequent during the busy period, and since each collision wastes the packet transmission, long preambles also increase the collision penalty. Previous work has described how LPL can shift to fully active mode after one node wakes up the network with a long-preamble packet [18]. Such an approach may pay a heavy listen price while nodes listen for potential follow-on traffic.

SCP-MAC eliminates long preambles, so its energy performance is not sensitive to varying traffic loads. Continuing the example of a target tracking application, we would configure SCP to run at a very low duty cycle to match the dominant time spent mostly idle. However, at times of heavy traffic, each hop in a scheduled MAC potentially adds additional latency and reduces throughput [13]. To reduce such multi-hop latency, we next develop *adaptive channel polling*. The basic idea is to detect bursty traffic and dynamically add additional, high-frequency polling slots to nodes on the path, allowing them to steam packets quickly over multiple hops. Unlike prior approaches (adaptive listen [27] and FRTS [24]), our approach works over every hop on the path. Unlike B-MAC optimizations [17] and fast-path [13], this approach requires no explicit signaling.

We describe adaptive channel polling with an example shown in Figure 2 where nodes transfer data over several hops. For the first part of the algorithm, consider that node A sends to node B. When B receives a packet during the first regular polling, it adds n high-frequency polls in the same frame, immediately following its regular poll. (We define n below.) If A has more packets to send, it sends them in these adaptive polling times. Spacing of adaptive slots is determined by the longest packet length that the physical layer supports.

If B finds none of its additional polls useful (such as if A had no additional data to send) it transitions back to its reg-

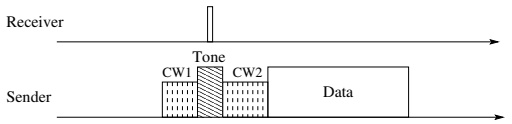


Figure 3. Two-phase contention in SCP-MAC.

ular, low-duty cycle polling period. If any of B's additional slots were successfully used, B extends the adaptive polling with n slots, allowing A to continue transmission. It also automatically plans an additional n in the next frame. If the duration of the traffic burst is shorter than one regular polling period, such process repeats at each hop, and the whole data can be transferred over N hops using N polling periods.

However, in many cases the burst of traffic lasts much longer than one regular polling period. We want to quickly bring all nodes on the path into the high-rate polling mode, and keep them in this mode until the burst ends. In this way, data can be quickly streamed from the source to the sink. We continue with the above example to illustrate the details. Due to the long burst, node A keeps sending to node B with frequent polling until the end of the first regular polling period. So far, the next-hop node C is not aware that A and B have begun adaptive transmission and polling, and C still follows the regular polling schedule. In order to shift node C quickly to adaptive polling, node A intentionally gives up the transmission opportunity in the second regular polling slot, allowing B to send to C without contention from A (as shown in Figure 2). When node C receives this packet, it too will shift to adaptive polling. In the following adaptive slots, both A and B contend to send. The same procedure repeats, and node D will start adaptive polling after the third regular polling slot. Eventually all nodes on the path will operate in the adaptive polling mode, and data can stream to the sink quickly using all-adaptive slots.

Finally, we consider what n should be—how many polls should be added. In a multi-hop network, each node contents with its previous- and next-hop nodes if they all have data to send. Thus, at full occupancy, each of the three nodes needs a slot to send, so that packets can quickly proceed downstream. We therefore set the number of adaptive polling slots n to 3.

In summary, adaptive polling slots are dynamically added and extended as driven by traffic. Regular polling slots are always reserved for new nodes to enter the high-rate polling and transmission quickly, reducing channel capturing by any single node. Multi-hop streaming reduces latency as well as buffer requirement at each node by quickly moving data over multiple hops.

2.3 Other Optimizations

On top of the basic scheme of scheduled and adaptive channel polling, SCP-MAC includes several other optimizations to improve its performance in handling collisions and overhearing.

2.3.1 Two-Phase Contention

Transmitting a packet in SCP-MAC involves two steps. First, the sender transmits a short wakeup tone timed to intersect with the receiver's channel polling. After waking up the receiver, the sender transmits the actual data packet. These two steps allow a *two-phase* contention procedure as shown in Figure 3. Before sending the tone, a node performs carrier sense by randomly select a slot within the first contention window (CW1). An idle channel allows the node to proceed, sending the wakeup tone that covers the rest of the contention window until the end of the receiver's polling time. If the node detects a busy channel (due to another node sending a tone first), it aborts transmission until the next frame and instead waits to receive the incoming packet. Only nodes that successfully send wakeup tones will enter the second contention window (CW2 in Figure 3). If such a node still detects channel idle in the second contention phase, it starts sending data.

The two-phase contention lowers the collision probability compared to a single contention period of equal duration, as only nodes that succeed in the first phase will enter into the second phase. Suppose we have m slots in a single contention window, the collision probability is roughly proportional to $1/m$. If we split the window into two separate periods, each with $m/2$ slots, the collision probability probability will become proportional to $4/m^2$. Therefore when $m > 4$, two-phased contention has lower probability of collision than a single period. (Alternatively, we can use fewer total contention slots to save energy and achieve the same collision performance.)

The reason that we can split the contention with fewer slots is that SCP tolerates collisions on tone transmissions—the wakeup tone only indicates network activity, not actual data. Thus, we can use a very small contention window for phase one. After phase one, only surviving nodes contend in the second phase. With fewer competing nodes, the collision probability on data transmission can be largely reduced. Our current implementation defaults to use 8 and 16 slots for tone and data contention windows, respectively.

2.3.2 Overhearing Avoidance Based on Headers

Unnecessary overhearing (receiving and then discarding packets addressed to other nodes) can be a large energy cost in high-density networks. In previous work, overhearing avoidance has been handled with a separate control channel [22] or RTS/CTS packets [26]. The use of RTS/CTS in SCP-MAC is optional and user configurable. When RTS/CTS is enabled, overhearing avoidance is performed the same way as that in S-MAC [26]. When RTS/CTS is disabled, SCP-MAC performs overhearing avoidance from MAC headers alone. To do this, a receiver examines the destination address of a packet immediately after receiving its MAC header, before completely receiving the packet. If it is a unicast packet destined to another node, it immediately stops the reception and places the radio to sleep.

We do not put checksums in the header, so the destination address can be corrupted when we read it. If it has no error, a correct decision can be made. If it is corrupted, the packet will be discarded anyway, so it is even more helpful to stop receiving the packet early.

It should be noted that overhearing avoidance has limited benefit for LPL, since there a decision can only be made after receiving the LPL preamble and the header. On average half the preamble must be received before making a decision, and the LPL preamble is much longer than the real packet.

3 Lower Bound of Energy Performance with Periodic Traffic

This section analyzes the energy performance of LPL and SCP. Our analysis is based on a single-hop network model where each node periodically generates packets at a fixed interval. Although somewhat artificial, the model roughly represents an environmental monitoring application where sensors are periodically sampled. Based on the model, we derive the lower bounds of energy consumption for both LPL and SCP. As a result, we also find the optimal operating parameters for these protocols to achieve their best performance.

3.1 Models and Metrics

Consider a network of $n+1$ nodes, where all nodes can hear each other directly, so each node has n neighbors. Each node generates one data packet at a regular interval T_{data}. Here we consider broadcast traffic (SCP performance is better with unicast traffic where overhearing avoidance is possible, as shown experimentally in Figure 12). Our analysis focuses on the energy consumption by the radio, and we do not model other components, such as the CPU or sensors. There are four stable radio states: transmitting, receiving, listening, and sleeping; each draws the power (energy per unit time) of P_{tx}, P_{rx}, P_{listen} and P_{sleep} respectively. Channel polling is different than normal listening, as it includes the time that the radio transitions from sleep to listen (typically around 2ms) and the brief sampling time to detect channel activity. We denote polling duration as t_{p1}, and its average power consumption as P_{poll}. We ignore radio transition costs for other states, assuming the on periods for data transfer are long enough to render transition costs negligible.

Both LPL and SCP are contention-based MACs, so transmission happens after carrier sense. To simplify the analysis, this section assumes that there is only one contention phase in SCP, and its contention window size is the same as that of the LPL. (This assumption favors LPL for reasons given in Section 2.3.1.) We denote the average time in carrier sense as t_{cs}. After carrier sense, a node first sends a wake-up tone and then the real packet. Each packet has a short, fixed preamble and a start symbol to synchronize the transmitter and receiver, and we included them in the packet length L_{data} for simplicity. The energy consumption of the radio is determined by how much time it spends in transmitting, receiving, listening, polling and sleeping, denoted as t_{tx}, t_{rx}, t_{listen}, t_{poll} and t_{sleep} respectively. In our analysis, all these time values are normalized to one second. They represent the fractions of time in one second the node in different states. We refer to them as *expected* time.

Table 1 summarizes all of our terms and gives typical values for the Mica2 radio (Chipcon CC1000 [9]) and an IEEE 802.15.4 radio (Chipcon CC2420 [10]). For both LPL and SCP, the expected energy consumption, per node, is the sum of the expected energy spent in each state:

$$\begin{aligned} E &= E_{cs}+E_{tx}+E_{rx}+E_{poll}+E_{sleep} \\ &= P_{listen}t_{cs}+P_{tx}t_{tx}+P_{rx}t_{rx} \\ &\quad +P_{poll}t_{poll}+P_{sleep}t_{sleep} \end{aligned} \tag{1}$$

We next derive the expected energy consumption for both asynchronous and scheduled channel polling schemes.

Symbol	Meaning	CC1000	CC2420
P_{tx}	Power in transmitting	31.2mW	52.2mW
P_{rx}	Power in receiving	22.2mW	56.4mW
P_{listen}	Power in listening	22.2mW	56.4mW
P_{sleep}	Power in sleeping	3μW	3μW
P_{poll}	Power in channel polling	7.4mW	12.3mW
t_{p1}	Avg. time to poll channel	3ms	2.5ms
t_{cs1}	Avg. carrier sense time	7ms	2ms
t_B	Time to Tx/Rx a byte	416μs	32μs
T_p	Channel polling period	Varying	Varying
T_{data}	Data packet period	Varying	Varying
r_{data}	Data packet rate ($1/T_{data}$)	Varying	Varying
L_{data}	Data packet length	50B	50B
n	Number of neighbors	10	10

Table 1. Symbols used in radio energy analysis, and typical values for the Mica2 radio (CC1000) and an 802.15.4 radio (CC2420)

3.2 Asynchronous Channel Polling: LPL

In LPL, nodes wake up asynchronously. A sender wakes up a receiver by sending a long preamble before each packet. The duration of the preamble is at least the same as the polling interval T_p, and thus the preamble length is

$$L_{preamble} = T_p/t_B \tag{2}$$

where, t_B is the time needed to transmit or receive a byte.

Before sending each packet, a node performs carrier sense. Recall that the average carrier sense time is t_{cs1}. The expected time a node spends in carrier sense is

$$t_{cs} = t_{cs1}/T_{data} = t_{cs1}r_{data} \tag{3}$$

where r_{data} is the rate of sending data packets on each node. The expected time that a node is in transmitting state is

$$\begin{aligned} t_{tx} &= (L_{preamble}+L_{data})t_B r_{data} \\ &= (T_p+L_{data}t_B)r_{data} \end{aligned} \tag{4}$$

The second line in the above equation is due to (2).

As each node periodically generates packets at the same rate of r_{data}, a node will periodically receive n packets from its n neighbors. The average length of the received preamble for each packet is $T_p/2$ due to the asynchronous polling. The expected time in receiving state is

$$t_{rx} = n(T_p/2+L_{data}t_B)r_{data} \tag{5}$$

The expected polling time is

$$t_{poll} = t_{p1}/T_p \tag{6}$$

The expected time with the radio asleep is simply the inactive time:

$$t_{sleep} = 1-t_{cs}-t_{tx}-t_{rx}-t_{poll} \tag{7}$$

Substituting Equations (3)–(7) into (1) and using Equation (2), we obtain the energy consumption with asynchronous channel polling as

$$\begin{aligned} E_r &= (P_{listen}t_{cs1}+P_{tx}(T_p+t_{pkt})+nP_{rx}(T_p/2+t_{pkt}))r_{data} \\ &\quad +P_{poll}t_{p1}/T_p \\ &\quad +P_{sleep}(1-(t_{cs1}+(n/2+1)T_p+(n+1)t_{pkt})r_{data} \\ &\quad -t_{p1}/T_p) \end{aligned} \tag{8}$$

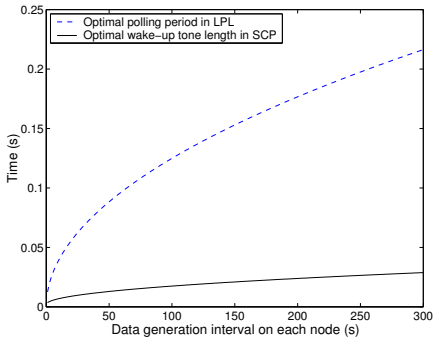


Figure 4. Optimal channel polling period in LPL (dotted), and wakeup-tone length in SCP (solid), given neighborhood size of 10.

Symbol	Meaning	Value
T_{sync}	SYNC packet period	Varying
r_{sync}	SYNC packet rate ($1/T_{sync}$)	Varying
L_{sync}	SYNC packet length	18B
L_{sB}	SYNC bytes piggybacked to data	2B
t_{mtone}	Minimum duration of wake-up tone	2ms

Table 2. Additional parameters in SCP-MAC

where t_{pkt} is the packet transmission time, and $t_{pkt} = L_{data} t_B$.

Assuming the packet length is fixed, we can see from Equation (8) that the energy consumption of a node changes with its neighborhood size n, data rate r_{data}, and channel polling period T_p. The tradeoff here is: reducing T_p reduces the cost of channel polling, but increases the energy spent in transmitting and receiving. This prompts the question: what is the optimal value of T_p to minimize the energy consumption, given a fixed n and r_{data}? We can obtain the answer by solving the following equation.

$$\frac{dE_r}{dT_p} = 0 \tag{9}$$

Substituting Equation (8) into (9), we find the optimal preamble length T_p for LPL is:

$$T^*_{p,r} = \sqrt{\frac{(P_{poll} - P_{sleep}) t_{p1}}{r_{data}(P_{tx} + nP_{rx}/2 - (n/2+1)P_{sleep})}} \tag{10}$$

In Figure 4, we show $T^*_{p,r}$ as a function of data rate by the dashed line using the parameters of the CC1000 radio listed in Table 1.

The optimal energy consumption with asynchronous channel polling is expressed by Equation (8) when $T_p = T^*_{p,r}$. We will show a numerical result in Section 3.3.3.

3.3 Scheduled Channel Polling: SCP

Next we turn to energy consumption for scheduled channel polling (SCP). In SCP, nodes synchronize their polling times with their neighbors, allowing shorter wake-up tones. However, SCP faces the additional cost of synchronization. Table 2 shows additional parameters in SCP (beyond those from Table 1).

Equation (1) still applies to SCP if we add the cost of maintaining synchronization. We explore this additional cost below, both with and without the ability to piggyback synchronization on data traffic. Before evaluating overall energy, we first investigate the tradeoff of wakeup tone length and synchronization frequency.

3.3.1 Synchronization Requirement and Tradeoffs

Several factors affect how we choose the wakeup tone length and the synchronization period, but the most important is the clock drift rate. Current CMOS crystal oscillators, such as those on the Berkeley motes, drift at a rate of 30–50 parts per million (ppm) [11]. To accommodate potential clock drift we extend the wake-up tone by a guard time.

Denote the synchronization period as T_{sync} (a configuration parameter) and the clock drift rate as r_{clk}. The maximum clock difference between two nodes is

$$t_{diff} = 2T_{sync} r_{clk} \tag{11}$$

where the factor of two reflects the worst case when each node's clock drifts in the opposite direction.

Since a sender does not know which direction its clock drifts with regard to a receiver, it needs to put the guard time in both directions, making the total guard time be $2t_{diff}$. If a node has n neighbors, each of them will send SYNC packets at the period of T_{sync}. Since every SYNC packet resynchronizes all nodes in the neighborhood, $(n+1)$ nodes effectively reduce the clock drift by $(n+1)$ times. Thus the guard time becomes

$$t_{guard} = 4T_{sync} r_{clk}/(n+1) \tag{12}$$

The duration of the wake-up tone is the guard time plus a short, fixed time

$$t_{tone} = 4T_{sync} r_{clk}/(n+1) + t_{mtone} \tag{13}$$

where t_{mtone} is the minimum time required to detect the tone. Since the time needed for the receiver to sample the channel (not including the radio transition time) and determine channel activity is around 0.5–2ms, depending on the radio speed, carrier-sense algorithm, and channel condition, we simply set $t_{mtone} = 2$ms for easy analysis.

There is a trade-off in determining T_{sync}: increasing T_{sync} reduces the energy cost of sending SYNC packets, but increases the cost on guard time and hence the wakeup tone length.

3.3.2 Best Case: Perfect Piggybacking

Given the fact that many types of data transmissions in sensor networks are periodic, synchronization information can be easily piggybacked on data. For example, all synchronization information can be piggybacked if $r_{data} \geq r_{sync}$. Here we investigate energy consumption for this case, assuming $r_{data} = r_{sync}$.

Since the data rate does not change, the expected carrier sense time is still expressed by Equation (3). The transmission time now is

$$t_{tx} = (t_{tone} + L_{sB} t_B + L_{data} t_B) r_{data} \tag{14}$$

Similarly, the reception time is

$$t_{rx} = n(t_{tone} + L_{sB} t_B + L_{data} t_B) r_{data} \tag{15}$$

The channel polling time and the sleep time can still be represented by Equations (6) and (7).

Substituting Equations (3), (14), (15), (6) and (7) into (1), we obtain the energy consumption of the scheduled channel polling with piggybacked synchronization as

$$\begin{aligned} E_{sp} &= P_{listen} t_{cs1} r_{data} \\ &\quad + (P_{tx} + nP_{rx})(t_{tone} + L_{sB} t_B + L_{data} t_B) r_{data} \\ &\quad + P_{poll} t_{p1} / T_p \\ &\quad + P_{sleep}[1 - t_{cs1} r_{data} \\ &\qquad - (n+1)(t_{tone} + L_{sB} t_B + L_{data} t_B) r_{data} \\ &\qquad - t_{p1} / T_p] \end{aligned} \tag{16}$$

Ideally, with the periodic traffic from all neighbors, a node should only poll the channel when there is a transmission from a neighbor. Thus the optimal polling period T_p for scheduled polling is

$$T^*_{p,sp} = \frac{1}{n(r_{data})} \tag{17}$$

The optimal energy consumption can be obtained by substituting Equation (13) into (16) and letting $T_p = T^*_{p,sp}$ and $T_{sync} = 1/r_{data}$. It is only a function of r_{data}. A numerical result will be shown in Section 3.3.3.

3.3.3 Worst Case: All Explicit Synchronization

Next we consider the worst case, assuming no piggybacking is possible and so all synchronization must be done with messages dedicated for that purpose. In effect, this increases the packet transmission rate by the value of SYNC packet rate, r_{sync}.

Since SYNC packets also require carrier sense, the expected time in carrier sense is

$$t_{cs} = t_{cs1}(r_{data} + r_{sync}) \tag{18}$$

where r_{data} is the data packet rate, and r_{sync} is the SYNC packet rate.

After carrier sense, a node first sends a wake-up tone to wake up the receiver and then sends the packet. The expected time in transmitting state is

$$t_{tx} = (t_{tone} + L_{data} t_B) r_{data} + (t_{tone} + L_{sync} t_B) r_{sync} \tag{19}$$

Compared with Equation (4), the long preamble is replaced with a short tone, but the packet rate is increased by r_{sync}.

Assuming all the data packets are broadcast (the worst case), the expected time in receiving is:

$$t_{rx} = n(t_{tone} + L_{data} t_B) r_{data} + n(t_{tone} + L_{data} t_B) r_{sync} \tag{20}$$

The expected time that a node polls the channel and sleep can still be expressed by Equations (6) and (7), respectively. But the values of t_{cs}, t_{tx} and t_{rx} in (7) are replaced by Equations (18)–(20).

Substituting Equations (18)–(20) and (6)–(7) into (1), we have the energy consumption in scheduled channel polling with independent SYNC packets as

$$\begin{aligned} E_{snp} &= P_{listen} t_{cs1}(r_{data} + r_{sync}) \\ &\quad + (P_{tx} + nP_{rx})(t_{tone} + L_{data} t_B) r_{data} \\ &\quad + (P_{tx} + nP_{rx})(t_{tone} + L_{sync} t_B) r_{sync} \end{aligned}$$

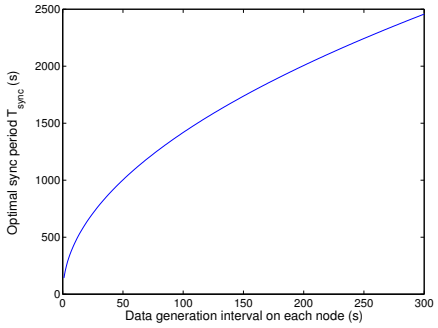


Figure 5. Optimal SYNC period for SCP-MAC.

$$\begin{aligned} &\quad + P_{poll} t_{p1} / T_p \\ &\quad + P_{sleep}[1 - t_{cs1}(r_{data} + r_{sync}) \\ &\qquad - (n+1)(t_{tone} + L_{data} t_B) r_{data} \\ &\qquad - (n+1)(t_{tone} + L_{sync} t_B) r_{sync} \\ &\qquad - t_{p1} / T_p] \end{aligned} \tag{21}$$

If we ignore the energy consumption in sleep state, the energy consumption with scheduled channel polling changes monotonically with the polling period T_p. The larger the T_p, the smaller the E_{snp}. This is different than the asynchronous channel polling as shown in Equation (8), since here the cost of sending and receiving a packet does not change with T_p. Ideally, with the periodic traffic from all neighbors, a node should only poll the channel when there is a transmission from a neighbor. Thus the optimal polling period for scheduled polling with independent SYNC packets is

$$T^*_{p,snp} = \frac{1}{n(r_{data} + r_{sync})} \tag{22}$$

Now we go back to the question "what is the optimal synchronization period T_{sync} that minimizes E_{snp}?" To answer the question, we substitute Equations (13) and (22) into (21), and solve the following equation

$$\frac{dE_{snp}}{dT_{sync}} = 0 \tag{23}$$

Thus the optimal T_{sync} is obtained as

$$T^*_{sync} = \sqrt{\frac{n(n+1)(E_l + P_t t_t + E_p)}{2 r_{data} r_{clk} P_t}} \tag{24}$$

where

$$\begin{aligned} E_l &= P_{listen} t_{cs1}, \\ P_t &= P_{tx} + nP_{rx} - (n+1)P_{sleep}, \\ t_t &= t_{mtone} + L_{sync} t_B, \\ E_p &= n(P_{poll} - P_{sleep}) t_{p1}. \end{aligned}$$

Once T^*_{sync} is known, we can obtain the optimal tone duration by substituting Equation (24) into (13), which is

$$t^*_{tone} = \frac{4 T^*_{sync} r_{clk}}{n+1} + t_{mtone} \tag{25}$$

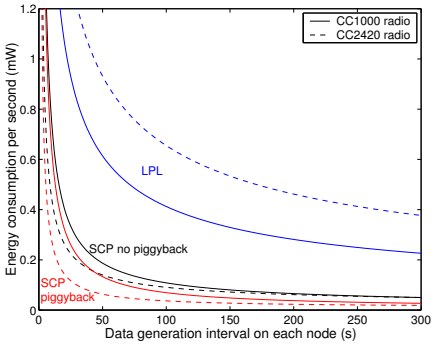


Figure 6. Analysis of optimal energy consumption for LPL and SCP with and without piggyback for CC1000 (solid lines) and CC2420 (dashed).

Figures 5 shows the optimal synchronization period. The optimal wake-up tone length is shown as the solid line in Figure 4.

From these results we can make several observations about how the parameters of a scheduled MAC compare to an unscheduled one. First, Figure 5 suggests that the clock synchronization can be quite rare, about 7× data transmission frequency during light loads ($T_{data} = 300$s) to $16 \times T_{data}$ during heavier loads ($T_{data} = 50$s). This observation suggests that synchronization overhead can be low. Second, clock synchronization and scheduled polling allows *much* shorter preambles than are possible with asynchronous media access. Finally, when piggybacking is used, synchronization happens "for free" on top of data, allowing much shorter tone lengths because of careful clock synchronization. The cost of piggybacking is also quite low, only 2 bytes per message.

The optimal energy consumption in scheduled channel polling with independent SYNC packets can be obtained by substituting Equations (22)–(25) into (21).

Figure 6 compares the optimal per-node energy consumption for the three cases we analyze (LPL, SCP with piggybacking of synchronization, and SCP without piggybacking). We consider two radio parameters: the CC1000 and the newer CC2420 implementing 802.15.4. We first observe that LPL consumes about 3–6 times more energy than SCP on the CC1000 radio. This cost is due to the expense of long preambles. We can also examine the benefits of piggybacking: it reduces the energy by about half when data is sent very rarely; the benefits are minimal when data is sent frequently because the cost of data packets then overwhelm control costs.

Finally, it is helpful to compare how these trends shift for newer generations of radios such as 802.15.4. The energy cost of SCP falls, because it takes shorter time to send data and perform carrier sense on the high-speed radio. However, the cost of LPL *increases*, because the preamble length still must be at least the length of the polling period, regardless of the radio speed. Since the higher-speed radio uses more power in transmit and receive, the overall energy usage in LPL increases. On CC2420, LPL consumes 8–15 times more energy than SCP.

The relative improvements for LPL and SCP are instructive, though. As an example, consider the 100 second interval between data packets. There SCP consumes 16% less energy (from 0.108mW to 0.091mW) than with a CC1000, while LPL consumes 59% *more* (from 0.413mW to 0.655mW). The reason for this is that 802.15.4 uses less energy per *byte*, but more energy per *time*. SCP sends its data quickly and is done, but LPL requires a long preamble for synchronization—this duration is only determined by the polling period, and *cannot* be reduced by newer, faster radios.

4 Protocol Implementation

We have implemented SCP-MAC in TinyOS [8] over the Mica2 motes [11] with the CC1000 radio. To provide a clean comparison of LPL and scheduling, we implement SCP as a layer over basic LPL. Here we describe this architecture, how it integrates with TinyOS, and details about piggybacking synchronization information. We also describe our preliminary port to MicaZ motes with the CC2420 radio supporting IEEE 802.15.4.

4.1 Software Architecture

We first describe the software architecture of SCP-MAC in TinyOS. Our implementation breaks MAC functionality into four layers (separate TinyOS components): the physical layer (PHY), a basic CSMA layer, the LPL layer, and the SCP layer. In addition to these modules, several parameters and options are configurable at compile time, including RTS/CTS handling, overhearing avoidance, and adaptive channel polling.

The PHY is at the bottom of the stack. It handles the radio states (sending, listening, receiving, sleeping, and warming up). It interacts directly with the radio, sending byte-by-byte with the Mica2, or packet-by-packet with the MicaZ. On reception, if necessary, it buffers all bytes from a packet and and passes the entire packet to the MAC when complete. It also implements and exports interfaces for physical carrier sense, transmission of the wakeup tone, CRC check, and time-stamping on transmitting and receiving of packets (to support time synchronization). To measure performance it can also record time spent in each radio state. The PHY module is designed to be MAC-independent and able to support contention-based or TDMA protocols, so it leaves backoff and similar functions to higher layers.

Above the PHY, we first implemented a basic CSMA protocol, providing a common service to both LPL and SCP. It includes preamble length as a parameter to packet transmission, allowing support for LPL. The CSMA layer is responsible for performing carrier sense and random backoff. For unicast traffic, it supports full RTS/CTS/DATA/ACK or simply DATA/ACK exchanges as a compile-time option. It also includes optional retransmission and overhearing avoidance.

LPL is implemented on top of the CSMA component. Its major purpose is to periodically poll the channel and send the radio to sleep when there is no activity. It adjusts preamble lengths on transmitted packets to ensure they intersect with polling frequency, and coordinates concurrent polling and transmission. To support SCP, LPL exports interfaces to query and adjust channel polling times.

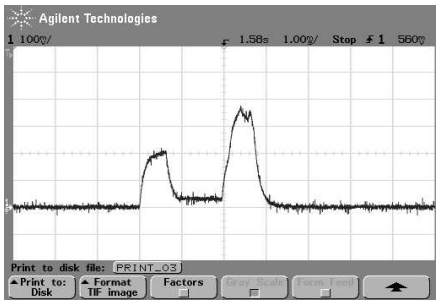


Figure 7. Channel polling process implemented in SCP-MAC.

Scheduling is implemented above the LPL module in the SCP module. It uses basic LPL to bootstrap schedules with SYNC packets. Once it has synchronized polling times with neighbor nodes, it switches to minimum preambles and wake-up tone transmission. It coordinates packet transmission timing to ensure short-duration wake-up tones are sent when neighbors are listening. It also implements the randomized contention window before wake-up tone transmission, which combines with CSMA-level contention for the data transmission to provide two independent contention periods. Finally, it includes a number of compile-time options, including SYNC piggybacking on broadcast data packets. (As future work we expect to also piggyback SYNC information in unicast exchanges.)

All three MAC components, CSMA, LPL and SCP, export the same interface for message transmission and reception. An application can easily switch MAC protocols by changing its component wiring. Such implementation promotes component reuse. This architecture also provides a common foundation for our performance evaluation in Section 5.

4.2 Interaction with TinyOS

Although we control radio activity, we depend on TinyOS for CPU power management. Our PHY layer coordinates with TinyOS to allow the CPU to sleep when the radio is not needed. Based on these components and the implementation from B-MAC we implemented the low-power channel polling. Figure 7 shows the current draw for channel polling captured by an oscilloscope (each x-axis tick is 1ms, each y-axis tick is 4mA). Our implementation provides similar results as the B-MAC implementation ([17], Figure 3).

We implemented a new timer in TinyOS to add support for dynamically adjusting timer values and asynchronous, low-jitter triggers. The synchronized channel polling in SCP-MAC requires to receive the timer firing events with very low jitter to minimize synchronization errors. Our timer implementation is based on the 8-bit hardware counter on Mica2. This timer runs independently from the CPU, allowing the CPU to sleep when no other activity is present. Because this timer uses an 8-bit counter running at 1024Hz, the timer overflows and must wake-up the CPU four times per second. We measured the energy cost of this event via an oscilloscope to confirm that its overhead is minimal compared to the cost of polling the radio (each timer event is about 0.4% the cost of a channel poll).

4.3 Efficient Piggybacking of Synchronization Information

To minimize the cost of synchronization we wish to avoid explicit SYNC packets. One SCP-MAC optimization is to piggyback synchronization information in broadcast packets. We are able to do so with no additional to packet length. Our normal MAC header includes 3 fields: packet type, source address and destination address. For broadcast data packets the destination address is normally set as the common broadcast address (0xFFFF) in TinyOS. However, the packet type field also redundantly indicates that the packet is a broadcast packet. We therefore use the type field to indicate broadcast packets and reuse the address field to piggyback schedule information.

On the receiver side, when SCP receives a broadcast data packet, it extracts piggybacked schedule information from the destination field, and performs schedule synchronization. It then replaces the destination field with the broadcast address before it passes the packet to its upper layer. Our approach piggybacks synchronization information onto broadcast packets for free, and it does not affect the operation of upper layers.

4.4 Port to IEEE 802.15.4 Radio

We have adapted SCP-MAC to run on the 802.15.4 radios found on the MicaZ hardware, in addition to it's original Mica2 platform with CC1000 radios. This port posed several interesting implementation challenges. To our knowledge, SCP-MAC is one of the first published energy-conserving MAC protocols for this radio. (Although B-MAC works on this radio, it currently operates without using LPL.)

There are a few important differences of the 802.15.4 radio (CC2420) to the CC1000, and some of them put significant challenges to implement LPL and SCP. First, CC2420 is a packet-level radio, and the microcontroller cannot get byte-level access. This potentially affects the accuracy of time synchronization. On the sender side, it must put a timestamp on the outgoing packet before it passes the packet to the radio. On the receiver side, the earliest time to take a timestamp is when the start symbol is detected. Such a delay is about 0.5ms as we have measured on the oscilloscope. It needs to be compensated for accurate synchronization.

Second, the radio automatically generates a preamble for each packet to comply with the 802.15.4 standard. CC2420 limits the preamble length to 16 bytes with a default length of 4 bytes. This is a strong challenge to implement the long LPL preambles, and also forces SCP to use a normal packet as the wakeup tone. A related problem is that the radio is much faster (250kb/s) than the CC1000 (19.2kb/s). At this speed, the longest packet (128B) consumes only about 4ms of airtime. Since our timer resolution is only 1024Hz, it requires some care to ensure that channel probes intersect this very short wakeup tone. To implement long preambles in LPL, we sequentially send multiple wakeup packets back to back. The major issue is to ensure that a receiver does not miss the "preamble" even if its channel polling time falls in a gap between the wakeup packets. To reduce these gaps, we pre-load the wakeup packet into the radio buffer before carrier sense, then resend the same packet from the buffer

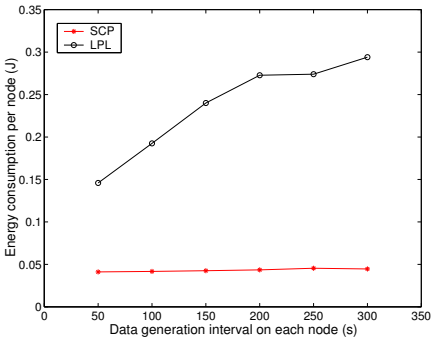


Figure 8. Mean energy consumption (J) for each node as traffic rate varies (assuming optimal configuration and periodic traffic).

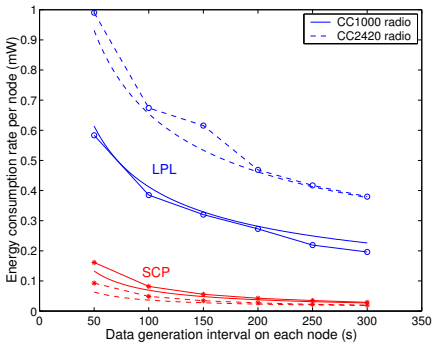


Figure 9. Mean energy consumption rate (J/s or W) for each node as traffic rate varies (assuming optimal configuration and periodic traffic). The radios are the CC1000 (solid lines) and CC2420 (dashed).

multiple times to make up a long preamble. This approach eliminates the buffer loading time of each wakeup packet, and effectively reduces the gap between two wakeup packets to about 30μs. Since the radio generates RSSI samples by averaging over a few recent symbols (128μs), the gap will not cause the radio to miss the preamble directly. In addition, our carrier sense algorithm automatically extends samples in case the first sample fails to give a clear answer. Our approach effectively allows arbitrarily long "preambles" and wakeup tones for LPL and SCP.

Our MicaZ implementation is still very preliminary. Although all layers of the stack work (PHY, CSMA, LPL, and SCP from Section 4.1), additional work is needed to make all layers fully functional. The LPL and SCP layers are sufficient to demonstrate the concepts and collect initial data, but significant work remains to tune the implementation and improve robustness.

5 Experimental Evaluation

The main contributions of this paper are to highlight the relative benefits of LPL and scheduling in energy conservation, and to propose a new MAC protocol, SCP-MAC, that optimally combines their strengths. We have implemented SCP-MAC to validate these contributions.

All actual MAC implementations have hundreds of specific design choices, many of which have effects on performance. Those details could distract us from the main question of comparing the advantages of scheduling on top of LPL. To control these details in comparing LPL and SCP, we use our own implementations of these protocols. This approach ensures that both LPL and SCP use the same basic components such as CSMA, physical-layer carrier sense, back-off, and other important parameters. In addition, it gives us the flexibility to compare SCP and LPL at a range of duty cycles and on both Mica2 and MicaZ hardware. (The current B-MAC implementation, as of March 31, 2006, does not support LPL on MicaZ.) We did validate that our basic channel polling is comparable to that in B-MAC (Section 4.2). Detailed comparison to complete implementations of other MACs such as B-MAC and WiseMAC is an area of future work. We do not compare to pure schedule-based protocols, such as S-MAC and T-MAC, since prior work [17] has shown that B-MAC outperforms these protocols.

5.1 Optimal Setup with Periodic Traffic

We first compare the energy performance of SCP and LPL under optimal configuration with completely periodic, known traffic. With static traffic loads we can optimize each for maximum energy conservation. We use our implementation to validate the analysis leading to Figure 6. To focus on this goal, we disabled advanced features in SCP-MAC, such as adaptive channel polling and overhearing avoidance.

MAC parameters vary based on network size and data rate. For this test we place 10 nodes in a single hop network. Each node periodically generates a 40B message (not including preamble) and broadcasts it to the network. For this test we consider very light traffic loads typical for very long-lived sensor networks: we vary each node's message generation interval from 50–300s. (Thus the aggregate data rate in the network is 1 message every 5–30s.)

For each static traffic load, we find out the optimal polling period of LPL and SCP from Equations (10) and (17). We run each experiment for 5 message periods, generating 50 total messages over each experiment. We report mean values of energy consumption per node. In this section we do not report standard deviations because they were small.

A control node begins and ends the experiment by broadcasting control packets to all nodes. We measure the energy consumption at each node by recording (in software) the time spent by the radio at different states. (We do not explicitly model CPU energy, but when the radio is off, the CPU is also in sleep mode except for timer maintenance, and we previously demonstrated that timer costs are negligible.) At the end of the experiment we collect this information from all nodes to a central measurement point and compute energy consumed by the radio.

Figure 8 shows the mean energy consumption of each node to send and receive all the messages. The energy consumption for SCP is almost constant at all rates, as the cost of sending each packet is about same. With broadcast traffic, all explicit SYNC packets are suppressed due to piggybacking. For LPL, the energy consumption increases at slower rates, since the optimal polling interval is longer (see Figure 4), therefore the cost on longer preambles is larger. In addition, the absolute cost of SCP is much lower than LPL: we can see

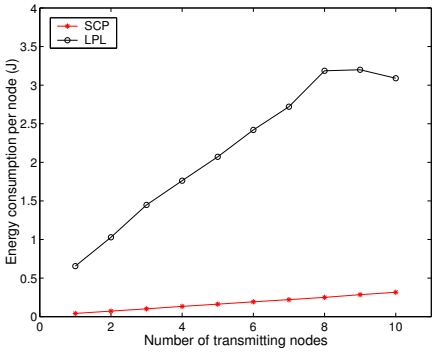


Figure 10. Energy consumptions on heavy traffic load with very low duty cycle configurations.

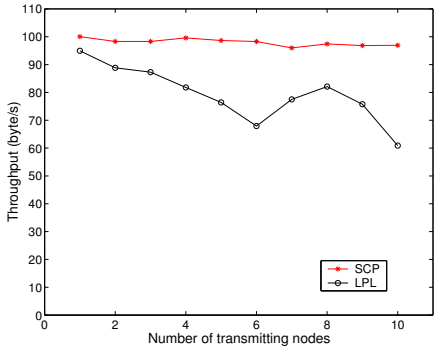


Figure 11. Throughput on heavy traffic load with very low duty cycle configurations.

that LPL requires 3–6 times more energy than SCP to send the same amount of data. This savings is because scheduling allows a much shorter wakeup tone on each data message.

We can also express the energy in terms of *rate*: Joules per second or Watts, as shown in Figure 9. (Figure 8 considered absolute energy required to send a fixed amount of data, per experiment, while Figure 9 normalizes this by experiment duration.) We expect slower traffic rates correspond to lower rates of energy consumption. Our experimental results indicate that LPL requires 3–6 times more power than SCP when both systems are optimized for periodic traffic. In addition, Figure 9 also shows our analytic results from Section 3.3.2, to compare analysis with experiment. We can see that both SCP and LPL experimental results very closely match their analytical results.

Figure 9 also shows preliminary experimental results for the MicaZ radio (CC2420). These values show means of only 3 runs for LPL and 5 for SCP. Again, the trends match analysis. More importantly, comparing the CC1000 costs to the CC2420 costs, the experimental results confirm that the energy consumption of LPL *increases* on the faster radio, while that of SCP *decreases*.

5.2 Performance with Unanticipated Traffic

In the prior section we consider optimal conditions for LPL and SCP with a completely known, periodic load. In many applications the traffic load is less predictable. For example, in tracking or monitoring applications such as fire detection in forests, there are long stretches of operation with no events, but then a detection causes a flurry of activity and very bursty traffic. Such a network *must* be optimized for the common case when nothing happens, yet it must also be able to handle bursty traffic when it occurs.

To evaluate these scenarios we next consider MAC performance when operating outside its optimal regime. We tune LPL and SCP for a 0.3% duty cycle, polling every second. Since the polling interval is the same for both MACs, energy draw without traffic is almost identical. (SCP requires slightly higher energy for schedule synchronization, but as discussed in Section 3.3.1, that cost is only 2% over polling.) Again, we have disabled adaptive polling and overhearing avoidance in SCP-MAC. All other parameters match our prior experiment.

To simulate the detection of a common event by multiple sensors, we trigger all nodes to enter "busy" mode at the same time. When busy, each node generates 20 100B-long messages, which are all broadcast messages. It sends as rapidly as possible, starting to send the next as soon as the prior message is transmitted. This burst of traffic exercises the network at an operating point different from its optimal.

To vary the degree of offered load, we vary the number of nodes in the network that start sending from 1 to 10. This traffic causes severe contention as the number of transmitting nodes increases in the network. Figure 10 shows the average energy consumption of each node as the number of transmitting nodes increases. We can see that at this heavy traffic, LPL consumes about 8 times more energy than SCP to transmit an equal amount of data. The main reason is the high cost of LPL preambles. When optimized for low duty cycle with a 1s polling interval, each packet sent by LPL includes a 1s preamble. SCP avoids this overhead.

Of course, one could extend LPL to shifting to shorter preambles for busy periods (as evaluated in [17] Section 6.3). However such a shift must be done conservatively to ensure all nodes agree to the transition—effectively a form of synchronization. B-MAC suggests that applications explicitly implement this optimization when appropriate, but in practice such optimization seems rarely used. While memory is dear in embedded processors, as they become more capable (while the expertise of programmers remains dear), providing greater capability in the basic protocol seems increasingly important.

With the same experiment, we have also quantified the benefits of two-phase contention in SCP-MAC by comparing its throughput with LPL. As described in Section 2, SCP uses *two* contention windows, one for the wakeup tone and the second for the data. Only nodes who successfully send tones will enter the second contention phase for sending data. LPL uses a single contention window of 32 slots here. To keep the total time spent contending identical, we divide the 32 slots into two 16-slot windows in SCP-MAC, rather than its usual 8 and 16-slot default contention window sizes.

As the offered load increases the contention algorithms of each protocol is stressed. With 10 transmitters, there is roughly a one-third chance of two nodes selecting the same slot and therefore colliding.

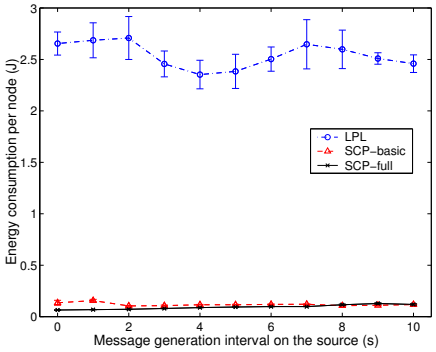


Figure 12. Mean energy consumption per node for multi-hop experiments (20 packets over 9 hops).

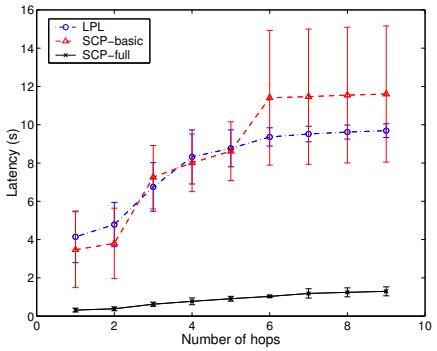


Figure 13. Mean packet latency over 9 hops at the heaviest load.

The two-phase contention window reduces overall collisions because even though there is a 10/16 (62%) chance of collision during the wakeup tone, collisions there do not matter since even multiple concurrent tones succeed in indicating the presence of traffic. Only nodes that succeed in the wakeup contention window will compete in the data contention period, thus it has only a $10/16^2$ (4%) effective collision rate.

We see the result of the more effective two-phase contention in Figure 11 as minimal reductions in SCP throughput as the number of transmitting nodes (and hence contention) increases. By comparison, LPL shows significantly lower throughput.

5.3 Performance in a Multi-hop Network

In this section, we evaluate the energy and latency performance of SCP-MAC in a multi-hop network. Here we consider varying traffic loads. In this case, there is no single best operating point for the MACs. Users often configure the MAC according to application requirements, perhaps based on how much latency the application can tolerate, or on the target operational lifetime. Here we configure the channel polling period to 1s for both LPL and SCP, providing a duty cycle of about 0.3%.

In this test we use a 9-hop linear network with 10 nodes. Since this is a multi-hop network we test SCP both with and without adaptive channel polling. Adaptive channel polling is designed to reduce latency with bursty traffic by adapting to streaming over multiple hops.

Data is sent from one edge of the network to the other. The source generates data with an inter-packet interval varying from 0–10s. Each test includes 20 packets sent at this rate, each 50B long. Nodes forward data long the line, with the most distant node receiving the packets. All packets are sent as unicast without RTS/CTS. We enable acknowledgments with up to three retries. Both LPL and SCP use header-based overhearing avoidance (Section 2.3.2), implemented in the CSMA module.

Figure 12 shows the mean energy consumption on each node for the entire experiment (sending 20 packets from the source to the sink). SCP without adaptive polling is labeled as SCP-basic, and SCP with adaptive polling is referred to as SCP-full. The results show that LPL consumes 20–40 times more energy than the full SCP-MAC. This difference is due to LPL's long preambles, both on reception of packets at each hop, and also due to reception by overhearers. Recall that, to decide that a packet is not relevant to a receiver, it must receive at least the header, including the preamble. Thus the cost of long preambles must be paid by not only intended recipients, but also *all* nodes within radio range.

A second cost of both SCP and LPL is false wakeups. When probing the channel, activity multiple hops away may be sufficient to positively indicate an active channel, even though the packet is not strong enough to be correctly received. While this cost must be paid by any MAC using channel polling, longer transmission times (due to preambles) in LPL mean that false wakeups occur more often in LPL relative to SCP. In addition, when a false wakeup happens, a node has to stay awake to receive the whole preamble and the erroneous packet. Therefore, LPL has a higher cost on each false wakeup than SCP.

Now we look at the latency of LPL and SCP over the 9-hop network. When traffic is light, we can expect that LPL and SCP have similar latency, as a packet can only go through one hop during one polling interval. We are more interested in verifying the performance of adaptive channel polling in the full SCP when heavy traffic occurs.

Figure 13 show the mean latency on each packet when the source generates the 20 packets at its fastest rate, sending the next immediately after the previous is done. The results show that the basic SCP and LPL have comparable latency. Essentially, there is a delay of 1s or more at each hop due to the fixed polling interval. In contrast, SCP with adaptive channel polling has a much lower latency—it is more than 7 times faster than LPL and the basic SCP in this test. Adaptive polling not only enables adjacent nodes to send multiple packets in one polling interval, it also enables multi-hop streaming. Essentially all nodes switch to high duty cycle polling after the first packet traverses the path.

Note that a similar result would be possible in LPL, provided the application was aware of this problem and took specific steps to control the preamble length (as described in [17] Section 6.3) and the latency. While in some cases the application designers may be able to do so, in many cases they may prefer to leave this detail to be done once in the MAC and selected automatically. In fact, Polastre et al. argue for modular system-level optimizations with SP [18].

6 Related Work

Energy-efficient MAC protocols have been a very active research area in wireless sensor networks. Existing work mainly focuses on two directions: TDMA and contention-based protocols. Two classes of contention-based protocols are those that add schedules and those that employ channel sampling. All protocols control radio duty cycle to avoid energy waste in idle listening; some also take approaches to avoid overhearing or add other optimizations.

Scheduled, contention-based protocols coordinate contention periods by synchronizing the wakeup schedules of nodes. The power-save mode in IEEE 802.11 [16] synchronizes wakeup times of nodes in a single-hop network. S-MAC [26, 27] developed a fully distributed algorithm to synchronize the wakeup schedules of nodes in a multi-hop network, and enables nodes to run at duty cycles of 1–10%. T-MAC [24] improves S-MAC by reducing the wakeup duration controlled by an adaptive timer. IEEE 802.15.4 proposes several power-saving techniques for when centralized controllers are present, but identifies synchronization in peer networks as out of scope of the standard. Polastre et al. proposed a simple synchronization scheme for 802.15.4 but did not attempt to optimize its performance [18].

The major advantage of scheduling is that a sender knows a receiver's wakeup time and thus transmits efficiently. However, the cost of listening for an entire contention interval is about ten times the cost of polling a channel for activity, thus the overhead in lightly used networks is higher than LPL-based approaches [17]. Moreover, the long listen interval limits these protocols to duty cycles of a few percent or more. Another potential drawback is the need to maintain schedules. This paper derives optimal parameters for scheduling under fixed traffic load, and that schedule coordination can be much less frequent than that is done currently.

Low-power listening is an approach where the channel is sampled very briefly for presence of activity rather than for specific data. Existing LPL protocols sample the channel in an uncoordinated fashion. STEM [21] explored the idea with a separate low-power paging channel. Hill [7] and El-Hoiydi [2] independently developed the approach of sending the wakeup signal by simply adding preambles in front of each transmitted packet. B-MAC [17] implemented LPL in TinyOS with a well-defined interface for applications to perform fine-grained control over the MAC. It also developed a new algorithm for clear channel assessment.

The major advantage of LPL is that it minimizes the overhead of listening time when there is no traffic. Since the channel polling cost is one-tenth the cost of listening for a contention interval in current scheduled protocols, LPL significantly reduces energy costs at light traffic loads. In addition, lacking the need to synchronize simplifies the LPL implementation and reduces code and memory size. However, there are a few major drawbacks of the above LPL algorithms. First, long preambles significantly increases the burden on transmitters. Such cost prevents the MAC from running at duty cycles less than a few percent, since at ultra-low duty cycles the send cost becomes prohibitive. Second, while LPL can be optimized for known, periodic traffic, its performance may significantly degrade at bursty and varying traffic loads. Finally, the simplicity of these approaches may be offset by additional complexity in applications to control their parameters.

Several groups have improved specific aspects of LPL. WiseMAC [3] can reduce the preamble length after an initial unicast packet with a long preamble. The receiver piggybacks its next polling time in the ACK packet, allowing the sender to send the next packet with a short preamble. This improvement has a few limitations. When a node sends broadcast packets, it has to use long preambles, even if it knows the polling times of all its neighbors. The reason is that nodes asynchronously poll the channel, and only long preambles ensure that all nodes can capture the packets. Due to clock drift, a node can only send the second packet with a short preamble if it closely follows the first one. A similar optimization is possible in B-MAC [17] for messages consisting of a sequence of packets, with similar limitations.

SP proposes a scheme for a node to piggyback its polling schedule when it sends a data packet with the long LPL preamble [18]. This allows neighbors to send back packets with short preambles at the node's next polling time. This idea is similar to WiseMAC, and has the same advantages and disadvantages as WiseMAC.

Uncertainty-driven B-MAC (UBMAC) [4] further reduces the long preambles in B-MAC by precisely estimating the clock drift between packet transmissions. Unfortunately, UBMAC's use of fine-grain time synchronization requires high-resolution timers, limiting the ability of the CPU to sleep and trading radio savings for CPU cost. In addition, since neighbors are not synchronized, broadcast still requires long preambles. We instead demonstrate that relatively coarse grain timers are sufficient, and provide a lower bound on energy consumption and optimal parameters based on traffic load and neighborhood size.

Halkes et al. studied the benefit of directly utilizing LPL in the listen intervals of S-MAC and T-MAC [5]. While this work demonstrated such benefits with simulation, it did not consider an optimal way to combine scheduling and LPL, and did not evaluate its scheme with real systems.

Our work differs from each of the above schemes to improve LPL in that we formally synchronize and maintain schedules, so that long preambles can be eliminated for all transmissions. In addition, we have several components that are completely novel, including quantifying the optimal performance analytically and evaluation of a working system with different traffic models. Our work quantitatively answers the fundamental question: how much benefit can be achieved when scheduling is added on top of LPL?

In addition to energy conservation, there has been work on reducing multi-hop latency. S-MAC uses RTS/CTS to inform the next-hop node of possible transmission. Theoretically it only reduces latency at every two hops, since CTS only reaches the next hop during the normal listen interval. T-MAC explores the similar idea with future-RTS packets. Other algorithms tries to explicitly stagger the wakeup time along a transmission path, such as D-MAC [14] and the fast path algorithm [13]. These protocols requires the help from the routing layer to obtain path information, so that they can precisely skew the wakeup times of nodes on the path. In

comparison, the adaptive channel polling in this paper enables multi-hop streaming at every hop, and it does not require any explicit signaling.

The second class of MAC protocols are based on TDMA. Traditional TDMA protocols often requires centralized control. For example, in LEACH [6] and BMA [12], nodes form clusters and cluster heads schedule transmissions. To extend the flexibility of TDMA, some distributed slot assignment schemes have been proposed, such as LMAC [25] and TRAMA [19]. Sohrabi and Pottie also proposed a protocol for distributed assignment of TDMA schedules [23]. Z-MAC [20] proposed a hybrid protocol to combine TDMA with CSMA. Z-MAC assigns each node a slot, but other nodes can borrow the slot (with contention) if its owner has no data to send. Z-MAC can significantly improves the throughput of B-MAC. However, its improvement on energy conservation is rather limited.

7 Conclusions

This paper proposes a new MAC protocol based on scheduled channel polling. By optimally combining scheduling and channel polling, it is able to operate sensor networks at duty cycles of 0.1% and lower. Through theoretical analysis we have derived a lower bound on energy consumption with periodic traffic, and found the best operating points for LPL and SCP. SCP-MAC can robustly handle bursty and varying traffic loads, and adaptive channel polling significantly reduces latency by enabling multi-hop streaming.

We have implemented SCP-MAC in TinyOS over both Mica2 and MicaZ motes. Our preliminary experiments show that SCP is able to achieve better energy performance than LPL by a factor of 3–6 with periodic traffic and more than 10 when traffic is bursty or varying. The relative performance of SCP improves on newer, faster radios like the CC2420, while that of LPL degrades.

Potential future work includes more thorough evaluation of SCP-MAC performance with different applications. Specifically, we plan to evaluate its scalability to networks size and density, its robustness under various network and traffic conditions, and to provide more complete comparisons with other MAC protocols. An additional potential benefit of scheduling to explore is how it can improve spatial reuse.

Acknowledgments

We would like to thank Yuan Li for her contributions to the SCP-MAC implementation in its early stage, and Joe Polastre for his help with CPU sleep support on Mica2 motes. We thank the anonymous SenSys reviewers and Anish Arora, our paper shepherd, for their comments about the paper.

Source code to SCP-MAC is available on the I-LENSE web site, `http://www.isi.edu/ilense/`.

8 References

[1] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. Habitat monitoring: Application driver for wireless communications technology. In *Proceedings of the ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, San Jose, Costa Rica, April 2001. ACM.

[2] A. El-Hoiydi. Spatial TDMA and CSMA with preamble sampling for low power ad hoc wireless sensor networks. In *Proceedings of the SeventhInternational Symposium on Computers and Communications (ISCC)*, pages 685–692, July 2002.

[3] A. El-Hoiydi, J.-D. Decotignie, C. Enz, and E. Le Roux. WiseMAC: An ultra low power MAC protocol for the wisenet wireless sensor networks (poster abstract). In *Proceedings of the First ACM SenSys Conference*, Los Angeles, CA, July 2003. November.

[4] Saurabh Ganeriwal, Deepak Ganesan, Hohyun Shim, Vlasios Tsiatsis, and Mani B. Srivastava. Estimating clock uncertainty for efficient duty-cycling in sensor networks. In *Proceedings of the 3rd ACM SenSys Conference*, pages 130–141, San Diego, CA, USA, November 2005. ACM.

[5] Gertjan Halkes, Tijs van Dam, and Koen Langendoen. Comparing energy-saving MAC protocols for wireless sensor networks. *MONET Special Issue on WLAN Optimization at the MAC and Network Levels*, (10):783–791, October 2005.

[6] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. In *Proceedings of the Hawaii International Conference on Systems Sciences*, January 2000.

[7] Jason Hill and David Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November 2002.

[8] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Cambridge, MA, USA, November 2000. ACM.

[9] Chipcon Inc. CC1000 data sheet. `http://www.chipcon.com/`.

[10] Chipcon Inc. CC2420 data sheet. `http://www.chipcon.com/`.

[11] Crossbow Technology Inc. Mica2 data sheet. `http://www.xbow.com/`.

[12] J. Li and G. Lazarou. A bit-map-assisted energy-efficient MAC scheme for wireless sensor networks. In *Proceedings of the 3rdInternational Symposium on Information Processing in Sensor Networks (IPSN)*, pages 55–60, Berkeley, CA, February 2004. April.

[13] Yuan Li, Wei Ye, and John Heidemann. Energy and latency control in low duty cycle MAC protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, New Orleans, LA, USA, March 2005.

[14] Gang Lu, Bhaskar Krishnamachari, and Cauligi Raghavendra. An adaptive energy-efficient and low-latency mac for data gathering in sensor networks. In *Workshop on Energy-Efficient Wireless Communications and Networks (EWCN '04), held in conjunction with the IEEE International Performance Computing and Communications Conference (IPCCC)*, April 2004.

[15] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, and David Culler. Wireless sensor networks for habitat monitoring. In *Proceedings of the ACM Workshop on Sensor Networks and Applications*, pages 88–97, Atlanta, Georgia, USA, September 2002. ACM.

[16] LAN MAN Standards Committee of the IEEE Computer Society. *Wireless LAN medium access control (MAC) and physical layer (PHY) specification*. IEEE, New York, NY, USA, IEEE Std 802.11-1999 edition, 1999.

[17] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd ACM SenSys Conference*, pages 95–107, Baltimore, MD, USA, November 2004. ACM.

[18] Joseph Polastre, Jonathan Hui, Philip Levis, Jerry Zhao, David Culler, Scott Shenker, and Ion Stoica. A unifying link absraction for wireless sensor networks. In *Proceedings of the Third ACM SenSys Conference*, pages 76–89, San Diego, California, USA, November 2005. ACM.

[19] Venkatesh Rajendran, Katia Obraczka, and J.J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. In *Proceedings of the First ACM SenSys Conference*, pages 181–193, Los Angeles, California, USA, November 2003. ACM.

[20] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min. Z-mac: a hybrid MAC for wireless sensor networks. In *Proceedings of the 3rd ACM SenSys Conference*, pages 90–101, San Diego, CA, USA, November 2005. ACM.

[21] Curt Schurgers, Vlasios Tsiatsis, Saurabh Ganeriwal, and Mani Srivastava. Optimizing sensor networks in the energy-latency-density design space. *IEEE Transactions on Mobile Computing*, 1(1):70–80, January 2002.

[22] S. Singh and C.S. Raghavendra. PAMAS: Power aware multi-access protocol with signalling for ad hoc networks. *ACM Computer Communication Review*, 28(3):5–26, July 1998.

[23] Katayoun Sohrabi and Gregory J. Pottie. Performance of a novel self-organization protocol for wireless ad hoc sensor networks. In *Proceedings of the IEEE 50th Vehicular Technology Conference*, pages 1222–1226, 1999.

[24] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the First ACM SenSys Conference*, pages 171–180, Los Angeles, California, USA, November 2003. ACM.

[25] L. van Hoesel and P. Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks. In *Proceedings of the 3rdInternational Symposium on Information Processing in Sensor Networks (IPSN)*, pages 55–60, Berkeley, CA, February 2004. April.

[26] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, NY, June 2002. IEEE.

[27] Wei Ye, John Heidemann, and Deborah Estrin. Medium access control with coordinated, adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 12(3):493–506, June 2004.