# Link Layer Support for Unified Radio Power Management in Wireless Sensor Networks

Kevin Klues, Guoliang Xing[*], Chenyang Lu
Department of Computer Science and Engineering
Washington University in St. Louis
{klueska, xing, lu}@cse.wustl.edu

## ABSTRACT

Radio power management is of paramount concern in wireless sensor networks that must achieve long lifetimes on scarce amounts of energy. While a multitude of power management protocols have been proposed in the past, the lack of system support for flexibly integrating them with a diverse set of applications and network platforms has made them difficult to use. Instead of proposing yet another power management protocol, this paper focuses on providing link layer support towards realizing a *Unified Power Management Architecture (UPMA)* for flexible radio power management in wireless sensor networks. In contrast to the monolithic approaches adopted by existing power management solutions, we provide (1) a set of standard interfaces that allow different power management protocols existing at the link layer to be easily implemented *on top of* common MAC level functionality, (2) an architectural framework for enabling these protocols to be easily swapped in and out depending on the needs of the applications that require them, and (3) a mechanism for coordinating the existence of multiple applications, each of which may have different requirements for the same underlying power management protocol. We have implemented these features on the Mica2 and Telosb radio stacks in TinyOS-2.0. Microbenchmark results demonstrate that the separation of power management from MAC level functionality incurs a negligible decrease in performance when compared to existing monolithic implementations. Two case studies show that the power management requirements of multiple applications can be easily coordinated, sometimes even resulting in better power savings than any one of them can achieve individually.

[*]Now with the Department of Computer Science, City University of Hong Kong

## Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols—*Protocol Architecture*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Modules and Interfaces*

## General Terms

Experimentation, Design

## Keywords

Wireless Sensor Networks, Radio Power Management, Architecture, Framework

## 1. INTRODUCTION

Energy is a scarce resource in many wireless sensor networks (WSNs). As wireless communication typically consumes a significant amount of energy, a tremendous amount of research has been dedicated to the development of radio power management protocols. While the goal of each protocol is ultimately the same (reduce the power consumed by the radio as much as possible), different protocols are better suited to some applications than others, depending on their workload characteristics and performance requirements.

Habitat monitoring applications [20][21], for example, have steady, periodic traffic and can benefit from power management protocols that duty cycle their radios in order to reduce energy consumption. Latencies in message delivery are tolerable, and sometimes even encouraged if it means that better power savings can be achieved. Applications such as intruder detection and tracking [10], on the other hand, rely on fast message delivery due to the urgency of the information they are providing. These applications are dominated by long periods of inactivity followed by large bursts of traffic once something has been detected. They would rather sacrifice some energy savings for an increased level of performance once they start generating messages.

As evidenced above, we cannot always use the same radio power management protocol for every type of application. A protocol optimized for low data rate, periodic habitat applications may perform poorly in an intruder detection application [27]. On the other hand, a protocol designed for the bursty workloads of an intruder detection application would be unnecessarily complex for a simple habitat monitoring application that always operates at a constant low rate. There must therefore be a way to allow an application to incorporate the use of whichever power management protocol is most appropriate for it.

Sometimes it may also be desirable to allow multiple applications to run concurrently on a single node. Imagine a set of habitat monitoring applications that each take readings from different sensors at different sampling rates. If each of these applications were to be installed on a single node, we would need a way of resolving the different requirements they impose on an underlying radio power management protocol. Sometimes it may be sufficient to configure a single protocol accordingly (either at compile time for static requirements or at runtime for dynamically changing requirements). Other times it may be more convenient to choose multiple protocols from among a set of predefined ones and find a way to compose them together in some coherent way. Unfortunately, despite significant progress in the development of each of these protocols, existing WSN systems still lack architectural support for flexible radio power management.

To address the issues outlined above, we propose an architecture that not only allows different radio power management protocols to be *flexibly* integrated into a fully functional wireless sensor network system, but also allows the requirements imposed by multiple applications to be *coordinated* in such a way that a single coherent radio power management solution is produced. While some power management protocols may exist at different layers in the networking protocol stack, we focus our current work on duty cycling protocols existing only at the data link layer. A high-level overview of an overall *Unified radio Power Management Architecture (UPMA)* is presented in [12].

Specifically, this paper makes the following primary contributions: (1) We design and implement a set of interfaces that allow different power management protocols to share common MAC functionality at the data link layer. Traditionally, MAC protocols and power management protocols have been developed together to produce a single monolithic implementation. Development of a new power management protocol often meant redesigning an entire radio stack from the ground up. The separation we propose gives different applications the ability to incorporate the use of whichever power management protocol is best suited to its needs, independent of the underlying MAC protocol it relies on. Certain power management optimizations (such as overhearing avoidance) may still need to be implemented *within* the MAC layer itself, but features such as this are beneficial to all power management protocols existing at the link layer, and their existence does not diminish the need for providing the separation that we do. (2) We propose an architectural framework that gives multiple applications the ability to specify differing requirements to a component implementing a single underlying radio power management protocol. Coordination of these requirements is achieved through a customizable component whose implementation depends on both the power management protocol in use and the types of applications running on top of it. (3) We demonstrate the practicality of this architecture by implementing it on top of both the Mica2 and Telosb radio stacks in TinyOS-2.0. We show that separating power management protocols from the MAC level functionality on which they rely increases flexibility while introducing only a negligible performance penalty. (4) Finally, we provide two case studies demonstrating how differing requirements from multiple applications can be resolved within our architecture. In each case study we show how to coordinate these requirements in a different way.

The rest of this paper is organized as follows. Section 2 provides an overview of the types of power management protocols this architecture is designed to support. Section 3 presents the design of the architecture as well as example implementations demonstrating its various features. Section 4 presents experimental results of the overhead incurred by implementing power management protocols using this architecture, as well results obtained during our two case studies. Finally, Section 5 concludes the paper.

## 2. POWER MANAGEMENT APPROACHES

In this section, we review existing approaches to radio power management in WSNs. This review provides the basis for our design of a common set of interfaces and a unified architecture that can support diverse power management protocols.

Existing approaches to radio power management fall into two categories: transmission power control and duty cycling. Transmission power control [19] reduces the energy consumed during communication by adjusting the power at which a radio transmits. Duty cycling reduces energy wasted during idle listening by allowing the radio to cycle between periods of activity and sleep. The architecture presented in this paper only focuses on supporting duty cycling protocols, which have proven to be a very efficient means of extending the system lifetime of WSNs. It can be accomplished by following one of three different approaches: TDMA, scheduled contention, or channel polling [27].

In TDMA based protocols, time is divided up into discrete time slots and allocated to all nodes within transmission range of one another. Nodes transmit during the time slots that have been allocated to them, and listen during the time slots that have been allocated to those nodes from which they wish to receive. When not transmitting or receiving, a node is free to sleep. Several different TDMA based protocols have been proposed for use in WSNs. These protocols include TRAMA [16], and DRAND [18]. One limitation of these types of protocols is that their schedules can be very sensitive to changes in network traffic or network topology, and all nodes sharing a schedule must remain synchronized with one another. Whenever one of these properties changes, a new TDMA schedule must somehow be generated and distributed to some subset of nodes in the network.

Protocols based on scheduled contention allow nodes to schedule times in which they will all be awake in order to communicate. Nodes within transmission range of one another synchronize their schedules to ensure that they are all awake at the same time. While awake, nodes contend for use of the radio channel through a process known as CSMA/CA. Nodes that gain access to the channel during this contention period are allowed to send, while all other nodes listen. Several energy-efficient MAC protocols based on sleep scheduling include S-MAC [26], T-MAC [23], and Z-MAC [17]. Despite the energy savings achieved using protocols of this type, energy is still wasted in maintaining synchronization and waking up when there is no data to transmit or receive.

Protocols based on channel polling (such as the Low Power Listening features of B-MAC [14], WiseMac [7] and X-MAC [3]) do not require synchronized contention periods for reception and transmission. All nodes independently wake up to poll the radio channel for activity. If there is, they prepare themselves for message reception. If there is not, they return immediately to sleep. Transmitting nodes send a stream of preamble bytes (or wake up tones) equal to the polling period of their destination nodes, in order to ensure that they wake up in time to receive any actual data. In lightly loaded networks, these types of protocols can achieve better power savings than scheduling based protocols since the overhead associated with long contention periods and synchronization is avoided. In heavily loaded networks, however, the overhead of transmitting a long stream of preamble bytes starts to outweigh these benefits.

Hybrid protocols (SCP [27], Funneling-MAC [1], and IEEE 802.15.4 [11]) combine some of the features present in TDMA, scheduled contention, and channel polling based protocols in order to find a reasonable tradeoff between message latency and power consumption. SCP, for example, combines the advantages provided by channel polling with those of scheduled contention in order to avoid the problems normally associated with requiring time synchronization while at the same time avoiding long preamble costs. Funneling-MAC, on the other hand, allows some nodes near a sink to run TDMA schedules while all others follow either a scheduled contention or polling based duty cycle. The overhead of maintaining the TDMA schedule is mitigated by the fact that only a small number of nodes actually need to follow it.

To improve the performance of duty cycling protocols, *backbone* based power management protocols have also been proposed to support performance sensitive applications. Protocols such as ASCENT [4], SPAN [5], GAF [24], and PEAS [25] set up a backbone of nodes that must be continually active in order to quickly forward messages between a source and its sink. Nodes not in the backbone are allowed to follow one of the four different types of duty cycling protocols presented in this section. Membership in the backbone is periodically changed in order to balance the power consumed by all nodes in the network.

## 3. DESIGN OF THE ARCHITECTURE

In this section, we describe the architectural support necessary for providing flexible radio power management at the data link layer. This architecture defines a set of interfaces that allow different duty cycling protocols to be independently implemented on top of a common underlying radio stack. The introduction of these interfaces enables support for flexibly integrating different duty cycling protocols into a system based on its specific application requirements. This architecture also includes support for defining components that are capable of coordinating the duty cycling requirements from multiple applications. By combining these requirements inside a separate architectural component, different duty cycling protocols can be made to work with a diverse set of applications and platforms.

### 3.1 Supporting Flexibility

We begin by defining a clear separation between what functionality is shared among all duty cycling protocols and what is not. We define this separation to occur above the features present in radio specific MAC implementations, but below the features distinguishing one duty cycling protocol from another. Figure 1 shows how we propose to perform this separation by introducing a set of uniform interfaces between duty cycling protocols and the MAC layer.

Three interfaces are made available through the MAC layer for use by different duty cycling protocols. Ideally, all radio stack implementations should provide each of these interfaces. If a particular MAC layer is unable to provide one of them, it may be limited in the types of duty cycling protocols that can be built on top of it. Each interface is described in greater detail below.

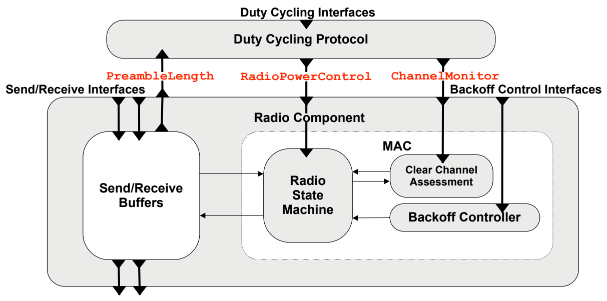

**Figure 1: Proposed separation of duty cycling protocols from common MAC level functionality**

**The `RadioPowerControl` Interface:**
The first of these interfaces is `RadioPowerControl`. This interface allows a radio to be switched between its *active* and *sleep* power states. It must be implemented by all types of MAC protocols, since without it, duty cycling of the radio is not possible.

```
interface RadioPowerControl {
  async command void on();
  async event void onDone(error_t error);
  async command void off();
  async event void offDone(error_t error);
}
```

**The `ChannelMonitor` Interface:**
The second interface is the `ChannelMonitor` interface. This interface is used to expose clear channel assessment (CCA) capabilites of a radio. This capability is required by all duty cycling protocols based on channel polling in order to determine if a radio channel has any activity on it or not. If this interface is not exposed, the use of certain duty cycling protocols (such as Low Power Listening) will not be possible.

```
interface ChannelMonitor {
  command void check();
  async event void free();
  async event void busy();
  event void error();
}
```

**The `PreambleLength` Interface:**
The third and final interface is the `PreambleLength` interface. This interface allows a duty cycling protocol to dynamically change the length of the preamble associated with a particular outgoing packet. It is intended for duty cycling protocols that require a constant wakeup tone of a certain length to be sent (transparent to the operation of the duty ccling protocol itself). For certain types of radios (in particular, packet radios like the cc2420 [6]), the implementation of this interface may need to be emulated by sending multiple packets with very short gaps instead of a constant stream

of preamble bytes. For duty cycling protocols that actually rely on the transmission of short packets, this interface is not necessarily required.

```
interface PreambleLength {
  async command void set(uint16_t numBytes);
  async command uint16_t get();
}
```

In addition to the interfaces presented above, the MAC layer may also need to support time stamping for incoming and outgoing packets in order to facilitate the development of time synchronization protocols required by both scheduled contention and TDMA based protocols. Time stamp data can be appended as metadata to any packets passed up the radio stack through the Receive interface of the MAC layer. Duty cycling protocols that exploit the use of this information simply need to extract it from the packets as they are passed up the stack.

## 3.2 Supporting Multiple Applications

UPMA provides a framework for coordinating different power management requirements from multiple applications. For purposes of this discussion, an *application* refers to any component in the system that wishes to specify a set of power management requirements to a duty cycling protocol existing at the link layer. Our architectural framework provides a mediator that resolves any conflicts among these applications' stated power requirements.[1] Figure 2 illustrates the framework for coordination.

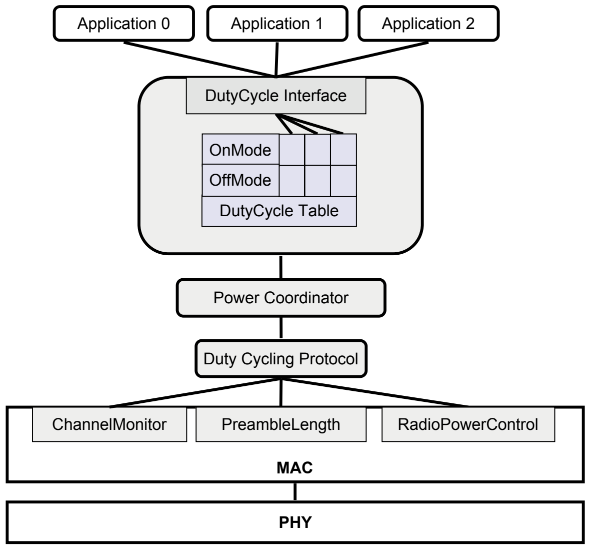

**Figure 2: Architecture for enabling the coordination of differing application requirements.**

Applications insert parameters into a *Power Management Table* using a set of predefined interfaces. The interfaces depend on the power management protocol in use. Protocols based on scheduled contention provide interfaces that allow applications to specify their on/off intervals, while protocols based on channel polling provide interfaces that allow them to specify a polling period and a preamble length. Protocols based on a hybrid of these two approaches require both sets of interfaces. Rows in the Power Management Table represent a single parameter type into which an application may supply a value. Columns are used to separate the values supplied by different components. For example, all parameters supplied by 'Application 1' in the figure will be stored in column 1, and all parameters supplied by 'Application 2' will be stored in column 2.

A *Power Coordinator* is used to coordinate the use of all parameters supplied to the Power Management Table by each application. It decides how to combine these parameters in order to provide a coherent radio power management solution that satisfies the needs of all applications as best it can. The Power Coordinator can be customized based on the requirements of the applications and the underlying duty cycling protocol on which they rely. We envision that a library of efficient standard coordination techniques (similar to design patterns) will be created that will be able to meet the needs of different types of applications.

## 3.3 Implementation

To enable the flexible integration of different radio power management protocols, the cc1000 and cc2420 radio stacks in TinyOS-2.x have been altered to expose the interfaces described in section 3.1. Implementations of several different duty cycling protocols have been created on top of these radio stacks for use on both the Mica2 and TelosB hardware platforms. We have chosen to use TinyOS-2.0 as our implementation platform since it is still maturing and does not yet have many radio power management protocols developed for it. Our hope is that as developers start porting implementations of their protocols from TinyOS-1.x to TinyOS-2.x, they will do so within the architecture presented here. Doing so will ease the development of these protocols and create a rich set components for use within the architecture.

In this section we describe sample implementations of both the polling based Low Power Listening (LPL) protocol and a simple scheduling based protocol called Simple Synchronous Sleeping (SSS). A third protocol that we call Basic Synchronous Sleeping (BSS) is also introduced. It is functionally similar to SSS, but differs in the type of interface it provides to its users. By providing implementations of both polling based and scheduling based duty cycling protocols, we are able to demonstrate the flexibility introduced by this new architecture.

We also present how two different coordination polices can be implemented using the mechanisms described in section 3.1. One policy coordinates the use of different duty cycling requirements from multiple applications, while the other coordinates a backbone maintenance protocol with these same duty cycling requirements.

We note that the implementations provided in this section are just examples that demonstrate the flexibility and efficacy of the architecture presented in this paper. More sophisticated duty cycling protocols and coordination policies can (and should) be implemented within the framework provided here. The focus of this work has not been to develop new protocols or coordination policies, but rather the architectural framework within which they can be developed.

[1]If only a single application happens to be running on the system, this mediator need not be included in its compilation.

### 3.3.1 Duty Cycling Protocols

LPL allows a radio to sleep for long periods of time, periodically polling the radio channel to check if there are any incoming packets. If no packet is present, it goes back to sleep for an amount of time equal to the node's polling interval. Packets are sent with preamble lengths equal to the size of the polling interval so that the destination node is guaranteed to be awake when the packet is sent. LPL allows a user to specify two different parameters: the time interval between subsequent checks for activity on the radio channel, and the preamble length for outgoing packets. We have implemented Low Power Listening on top of the interfaces presented in section 3.1.[2]

We have also designed and implemented a scheduling based duty cycling protocol known as SSS (Simple Synchronous Sleeping) on top of our interfaces. SSS relies on global time synchronization of all nodes in a network to precisely control their duty cycles. This protocol is intended as an example of how a scheduling based protocol *could* be implemented, rather than a replacement for more robust protocols. This protocol allows the duty cycle of the radio to be tuned through the following interface:

```
interface RadioDutyCycling {
  command error_t setDutyCycle(uint8_t on, uint8_t off);
  command error_t setOnTime(uint8_t onTime);
  command error_t setOffTime(uint8_t offTime);
  event void beginOnTime();
  event void beginOffTime();
}
```

A higher layer uses this interface to set the duty cycle of the radio and be notified whenever it has been switched on or off. Since the start of every radio's duty cycle must be synchronized, all nodes having the same duty cycle will be able to communicate with each other during the on time of the radio (using CSMA/CA) and conserve energy during the off time.

The third protocol we have implemented is BSS (Basic Synchronous Sleeping). Both SSS and BSS require time synchronization for all nodes in a network, and they both turn the radio on and off for certain time durations as specified by the user. The primary difference between the two is that SSS allows an application to specify a periodic radio duty cycle, while BSS requires that an application explicitly request the radio to be turned on or off just before making each transition. This allows the on and off periods to be changed on every transition rather than requiring them to be periodic. SSS could, in fact, be simulated using BSS by always specifying the same on and off times every time the appropriate transition was supposed to occur. Because of this flexibility in changing on and off times at every transition, BSS has the capability of supporting more sophisticated scheduling algorithms as well as integrating different schedules supplied by multiple applications.

An application can inform BSS when (as well as for how long) to power the radio on and off using the following interface:

[2]Due to the limited preamble lengths allowed by the cc2420 hardware, our specific implementation of LPL can only be used by the cc1000 radio stack. Ye, et al., [27] propose a way of implementing a variation of LPL for use on the cc2420 radio stack. This variant is compatible with our architecture and we plan on implementing it as future work.

```
interface DutyCycleTimes {
  command turnOnFor(uint32_t onTime);
  command turnOffFor(uint32_t offTime);
  event void ready();
}
```

Calling the `turnOnFor()` and `turnOffFor()` commands does not necessarily indicate that the radio will be turned on or off immediately. These commands are a way of specifying the on and off times that will be used the next time BSS signals the `ready` event. To duty cycle a radio, an application can alternate calls to `turnOnFor()` and `turnOffFor()` within the body of the `ready()` event. If no calls to `turnOnFor()` or `turnOffFor()` are made between subsequent `ready()` events, the power state of the radio remains unchanged and the next `ready()` event is signaled after the same amount of time it took to receive the previous `ready()` event.

Our current implementations of SSS and BSS use a simple time synchronization scheme that operates in a single-hop network. Under this scheme, all nodes other than the base station turn on their radio interfaces once they are booted, and leave it on until they have a received a synchronization packet. Once the base station is started, it sends the synchronization packet, and all nodes in the network start a timer simultaneously. There are no synchronization updates and clock drift is not taken into consideration. If one of the nodes happens to not hear the synchronization message, then it will not become synchronized and the entire network will need to be rebooted. It is left as future work to implement more sophisticated sychronization protocols using the time stamping features provided by the MAC layer [8, 9, 13].

In addition to the simple duty cycling protocols presented here, more complicated duty cycling protocols such as SCP-MAC and 802.15.4 can also be implemented within this architecture. We consider these two protocols as examples because 802.15.4 is the most widely accepted industry standard, and both 802.15.4 and SCP-MAC are both representative hybrid protocols that combine the features present in TDMA, scheduled contention, and polling based protocols. Both protocols would use the underlying CSMA/CA capabilities of the radio to contend for use of the radio channel, as well as the `RadioPowerControl` interface to turn the radio on and off. The time stamping feature provided by the MAC layer would also be used in order to provide a time synchronization service for use by the protocols. 802.15.4 would use the `RadioPowerControl` interface to turn on the radio for both its contention access period (CAP) and its contention free period (CFP). During the CAP it would use CSMA/CA to send and receive messages over the radio and during the CFP it would use a TDMA schedule to decide in which guaranteed times slots a node should be sending. After both the CAP and CFP, the radio would be turned off again via the `RadioPowerControl` interface. SCP would only use the `RadioPowerControl` interface to turn the radio on for the reception of packets, and off again once all packets were received. It would additionally require the `ChannelMonitor` interface to perform its channel polling capabilities and the `PreambleLength` interface to adjust the number of preamble bytes associated with any outgoing packets.

### 3.3.2 Coordination Policies

We have implemented two coordination policies that are able to combine different duty cycling requirements specified through the `RadioDutyCyling` interface. The parameters passed to this interface are stored in a *Power Management Table*, and a *Power Coordinator* is used to combine these requirements to produce a single coherent duty cycling schedule. This schedule is used to inform BSS of the on and off times it should use when duty cycling the radio. Note that different systems may desire different coordination policies. We provide the two coordination policies just as examples to demonstrate the capability of our architectural framework to coordinate multiple power management requirements.

In our first coordination policy, the Power Coordinator is implemented to aggregate the duty cycles specified by multiple applications according to an OR policy as shown in Figure 3.

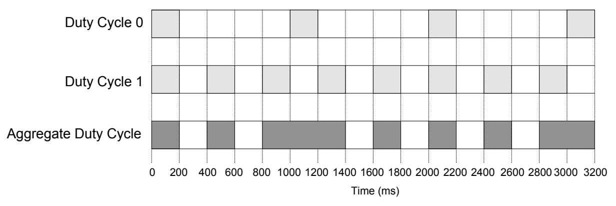

**Figure 3: Aggregation of multiple duty cycles**

This coordination policy is implemented as follows. (1) All duty cycles from all applications are synchronized to begin at the same time instant. (2) They all run periodically according to their own schedule. (3) If *any* of the duty cycles requires the radio to be on at any particular point in time, the radio will be turned on. (4) Only if *all* duty cycles indicate that the radio should be turned off will the radio ever be turned off. The coordinator creates a schedule whose length equals the least common multiple of the duty cycles of all applications.

In Figure 3 we see that 'Duty Cycle 0' has an on time duration of 200ms and an off time duration of 800ms, while 'Duty Cycle 1' has both an on and off time duration of 200ms. the period of 'Duty Cycle 0' is therefore 1000ms, while the period of 'Duty Cycle 1' is just 400ms. In order to find the period of the 'Aggregate Duty Cycle' schedule, the least common multiple of all duty cycles must be determined. In this case it is 2000ms. Since multiple on and off periods will exist within one period, BSS is more appropriate than SSS for executing the aggregated schedule.

The second coordination policy we have implemented allows a backbone based duty cycling protocol to be introduced as one of the applications specifying parameters to the `RadioDutyCyling` interface. The backbone based protocol we use is known as PEAS [25]. When nodes first wake up in a PEAS enabled network, they send out a probing message to determine if any of their neighboring nodes are awake and operating. If they do not hear any responses, they decide to become *active* and turn their radios on accordingly. Once a node has become active, it will remain active until its power supply has been depleted. Active nodes take on the the responsibility of responding to probing messages sent by inactive nodes. If inactive nodes hear one of these responses, they return immediately to sleep and wait some predetermined amount of time before sending out the next probe. The amount of time they have to wait changes dynamically based on the number of active nodes within their probing range as well as the frequency with which other inactive nodes send out their probing messages. We have implemented a lightweight version of PEAS that uses the same probe/reply mechanism as described above, but uses a fixed delay time between each probing message.

Our coordination policy allows PEAS to determine which nodes should be active, but instead of keeping these nodes always turned on, it allows them to run according to some duty cycle. The duty cycles on active nodes are aggregated according to the same OR policy on top of BSS as described before. Only active nodes run these aggregated duty cycles, sending and receiving both their own messages and PEAS probe/reply messages in the background. Inactive nodes do not run any applications and simply run a periodic sleep schedule specified by PEAS, continuously probing if they should become active or not.

We note that PEAS saves energy by controlling the spatial density of active nodes, while duty cycling saves energy by controlling the temporal frequency of radio activity. By combining the energy benefits provided by PEAS with those of duty cycling, this coordination policy is able to save more energy in a network than using of either one of these protocols individually. A key advantage of implementing all of these protocols within the architecture described in this paper is that neither the implementations of PEAS, BSS, nor any of the applications needs to be altered in order to achieve these energy savings. All coordination is handled by the implementation of the coordination policy itself.

## 4. EVALUATION

This section presents the empirical evaluation of our architecture. The first set of experiments evaluates the efficiency of the uniform interfaces between duty cycling protocols and the MAC in terms of both performance and code size. The second set of experiments evaluates the effectiveness of our framework in coordinating the duty cycle requirements of multiple applications. The final set of experiments evaluates the coordination of a backbone protocol and duty cycle requirements of multiple applications.

### 4.1 Efficiency

The first set of experiments involve comparing the monolithic implementation of Low Power Listening (LPL) on the cc1000 radio stack with the one designed for use within our architecture. Our experimental settings are the same as the ones presented for the full B-MAC implementation in [14]. We also compare the difference in the code size between the two implementations. By showing that our implementation of Low Power Listening is comparable to the original one in terms of both performance and code size, we are able to demonstrate that our architecture provides just as efficient a framework for implementing it as the original one. Since our implementation is not implemented within the MAC component of the default radio stack, however, it provides much more flexibility.

We first measure throughput vs. number of nodes in a single hop network. There is one receiver, with a variable number of senders from 1 to 4. All senders are equidistant from the receiver at 2 feet. Each sender transmits as often as possible with messages containing 38 bytes of data and 8 preamble bytes. We measure the total throughput (bits per second) at the receiver over 2 minutes. The results are given in Figure 4.

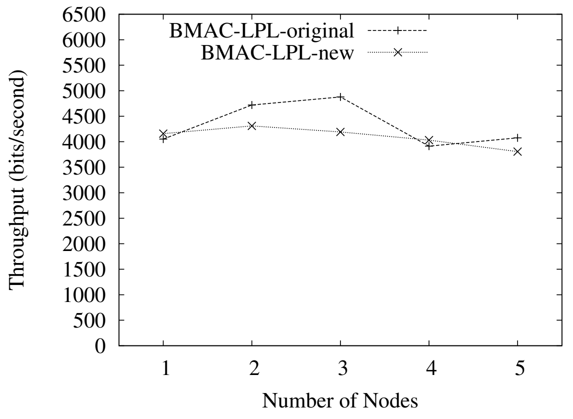

**Figure 4: Throughput vs. Number of nodes at 100% duty cycle for two different LPL implementations**

With only one node in the network, performance between the two implementations is almost identical. While the original LPL implementation achieves slightly higher throughput for two and three nodes, the results for the two implementations are comparable again for four and five.

Next, we measure the delivery latency vs. number of hops in a fixed route multi-hop network. Nodes are placed in a chain, with the first node being both the source and the sink node. Messages are sent from one node to the next until the last node in the chain is reached. Messages are then sent in reverse back to the original sender. The number of nodes varies from 2 to 5, resulting in 2, 4, 6, and 8 hops respectively. The sender sends 20 messages, each containing 38 bytes of payload and a variable number of preamble bytes depending on the length of the LPL check interval that has been selected. LPL check intervals of "always on", 800ms, and 1600ms were chosen, and the average latency from source to sink of each data packet was measured. The results are given in Figure 5. In this experiment, we see that the results for both LPL implementations are identical for all measured check intervals.

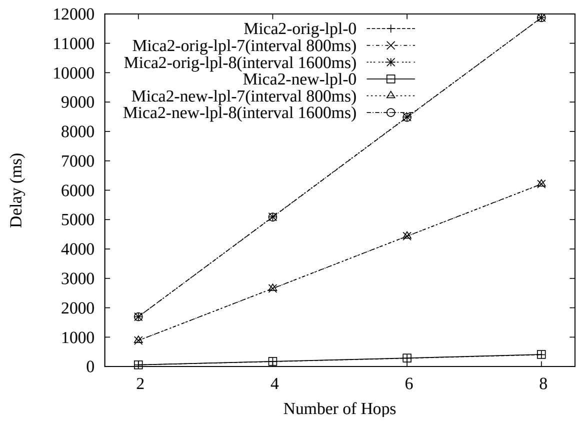

**Figure 5: Latency vs. Number of Hops at different check intervals for two different LPL implementations**

| | Original LPL | New LPL |
|---|---|---|
| | RAM/ROM | RAM/ROM |
| SenderApp | 383/11956 | 394/12350 |
| ReceiverApp | 705/15098 | 716/15560 |

**Table 1: LPL Memory Footprint (Bytes)**

Table 1 shows the difference in code size for each implementation when they are compiled into the applications used in the experiments above. As expected, both the RAM and ROM sizes for the new implementation are slightly larger than for original one. The main contributor to this increase in size is the extra timer required by the new LPL implementation. In the original implementation of LPL, the timer used to switch between the different states of the radio was shared by the LPL implementation. Other contributors include additional flags and logic needed to coordinate the use of the interfaces now provided by the radio stack.

The second set of experiments shows the performance characteristics of our SSS implementation. The results of these experiments show that it is easy to reuse the implementation of this sleep scheduling policy on top of two very different radio stack implementations. Results are given for both Mica2 and TelosB.

The setups for each experiment are the same as those used for LPL. For measuring throughput vs. number of nodes, we ran SSS at duty cycles of 100%, 47%, and 20%, and measured the total throughput for each duty cycle over 2 minutes. For measuring latency vs. number of hops, we ran SSS at a 50% duty cycle, and measured the average latency from source to sink for a single packet.

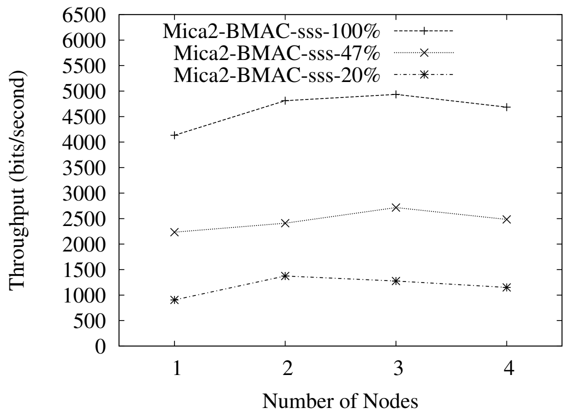

**Figure 6: Throughput vs. Number of nodes at different duty cycles for the SSS implementation on mica2**

Figures 6 and 7 show that SSS is able to deliver more data as its duty cycle is increased. As expected, TelosB achieves higher throughput for all duty cycles in both experiments because data is sent at a much higher rate by the cc2420 radio than by the cc1000 radio.

Figure 8 shows the latency associated with running SSS. Once again, the higher data rate of the cc2420 radio accounts for the difference in performance between the TelosB and the Mica2 platforms in this experiment.

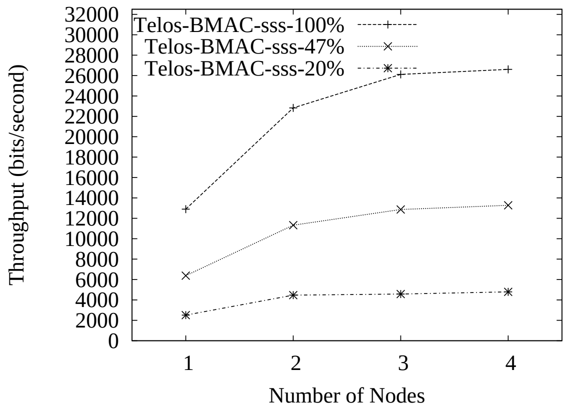

**Figure 7: Throughput vs. Number of nodes at different duty cycles for the SSS implementation on telosb**

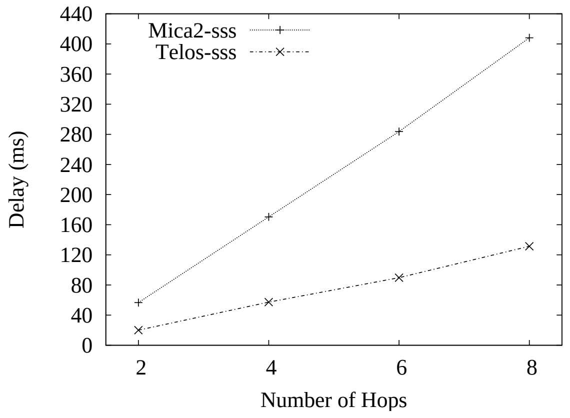

**Figure 8: Latency vs. Number of Hops at 50% duty cycle for the SSS implementation on both mica2 and telosb**

Overall, the results of these experiments show the following: (1) Implementing LPL using our framework only incurs a negligible performance penalty. (2) Exposing the proposed MAC layer interfaces may produce a slight increase in code size, but it allows much more flexibility when choosing the sleep scheduling policy that is most appropriate. (3) Both channel polling and scheduled contention based protocols can easily be implemented on top of these interfaces and used by different mote platforms. These implementations produce results typical of these types of protocols.

## 4.2 Coordinating Multiple Duty Cycles

In this section, we evaluate the architecture in terms of the different coordination policies used to combine duty cycles specified by multiple applications. The first experiment demonstrates the ability to combine the duty cycling requirements of multiple applications in a way that is transparent to each of them. The second one presents the results obtained from coordinating the duty cycling requirement from multiple applications for use with PEAS and BSS. The network used in this set of experiments is a one-hop cluster consisting of a master TelosB node and a number of slave TelosB nodes. Each slave node runs a sensing application that periodically sends packets to the master node.

Although each node only runs a single application, up to 6 different applications can be running in the network at any given time. The on time of the duty cycle for each application is 200ms, with off times of 200ms, 600ms, 1.4s, 3s, 6s, and 12.6s, respectively. Each application sends a packet of 66 bytes (including header and payload) at a random time within the 200ms active period of each duty cycle. The master node is able to receive packets from each application by running an aggregate duty cycle according to the OR policy described in section 3.2.

The first run of experiments consists of the master node and two slave nodes running the application with the lowest duty cycle. Two more slave nodes running the application with the next highest duty cycle are then added in each following run. Each run lasts for 320 s. Figure 9 shows the delivery ratio measured at the master node for each run. We can see that the delivery ratio remains close to 100% as the number of applications increases. Once all six applications (total 12 nodes) have been added to the network, however, we do begin to see a slight increase in the number of packets that are lost.

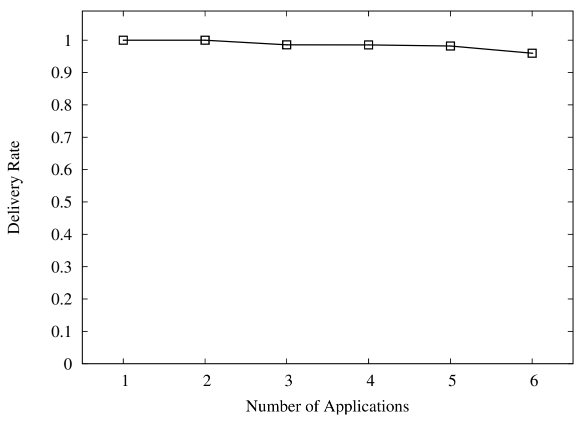

**Figure 9: The delivery ratio measured at the master node.**

Figure 10 shows the duty cycle measured at the master node. A 100% duty cycle corresponds to the radio always being on, and a 0% duty cycles corresponds to the radio always being off. The duty cycle was calculated by instrumenting the cc2420 radio stack with a 32 KHz timer in order to measure the amount of time spent in each radio state. We can see that the duty cycle measured at the master node matches the predicted curve, verifying the correctness of the combination logic of the aggregator. As a baseline we also show the predicted duty cycle of the master node if no aggregation policy were used. This duty cycle is simply calculated as the sum of the duty cycles of all applications in the network. As shown in Figure 10, performing the combination will always yield a lower overall duty cycle than not performing it.

The results in this section demonstrate that this coordination policy is capable of correctly combining the duty cycles specified by multiple applications, and that combining these duty cycles according to some aggregation policy can potentially lead to lower energy consumption.

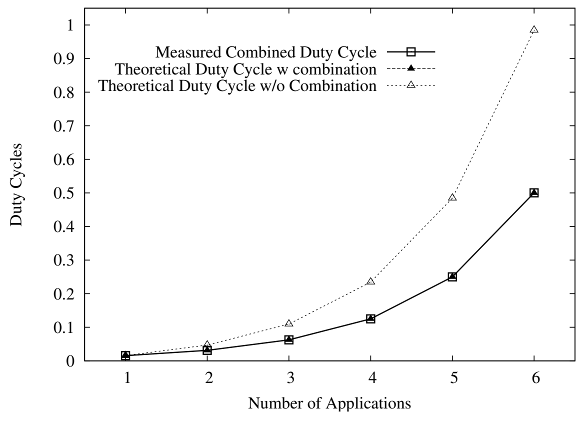

**Figure 10: The duty cycle of the master node.**

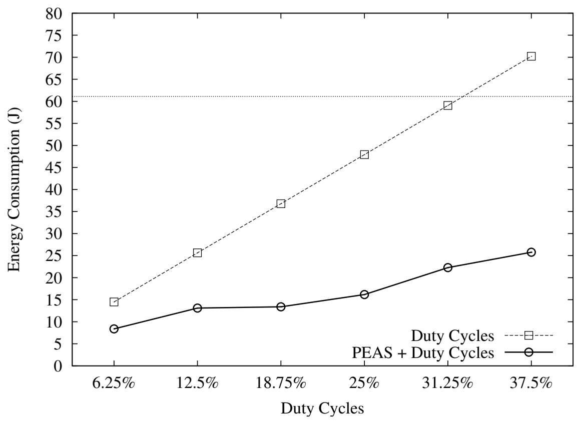

**Figure 11: The total energy consumption of the network.**

## 4.3 Coordinating Duty Cycles with PEAS

In this section, we evaluate the architecture in terms of the coordination policy allowing PEAS to be combined with applications specifying different duty cycles. The network consists of a master Telosb node and 15 slave Telosb nodes placed within a $5 \times 3$ grid. Each slave node runs both PEAS as well as one of six different applications. Each application runs with a duty cycle period of 3.2s. The on times of each duty cycle range from 200ms to 1.2s in steps of 200ms. Nodes designated as 'inactive' by PEAS run with a duty cycle period of 16s and an on time of 200ms. Inactive nodes send probe messages at some random time within their on periods, and active nodes send messages from their applications at some time during their on periods. Although all nodes are within communication range of one another, the probing range of PEAS is limited to 1.5 times the grid width.

In this set of experiments we measure the total energy consumed by the radio for all nodes in the network. The amount of energy used by each radio is measured as the sum of the energy consumed in each of four different radio states: idle, receiving, transmitting, and sleeping. We first measure the total time that the radio spends in each radio state by instrumenting the Telosb cc2420 radio stack with a 32 KHz timer. We then calculate the energy consumed in each state by multiplying the total time the radio spends in that state by the power consumed in that state. These power consumption values are all taken directly from the cc2420 data sheet [6][3].

As a baseline, we first measure the power consumption of the network when only PEAS is enabled and all active nodes have their radios powered all the time. We then disable PEAS and allow all nodes to run their duty cycling applications, without PEAS specifying certain nodes as inactive. Finally, we measure the power consumption when these two policies are combined: PEAS is enabled and only active nodes are able to run their application. Figure 11 shows the total energy consumption of the network in each situation. The baseline is represented by a straight line.

[3]There are two different sleeping modes available on the cc2420. In the sleeping mode benchmarked here, the transmitter is turned off while the crystal oscillator and voltage regulator remain on. In the data sheet this state is referred to as IDLE.

We can see from Figure 11 that the policy combining PEAS with each application duty cycle yields the lowest energy consumption. These energy savings are achieved by (1) allowing PEAS to choose the subset of nodes that will actually run each application, and (2) allowing those nodes chosen by PEAS to run at their application-specified duty cycle. Overall, the energy consumption under the combined policy is $57-86\%$ lower than running PEAS alone and $42-63\%$ lower than duty cycling alone.

The results in this section demonstrate the power of combining complementary power management protocols using the architectural framework provided in this paper. By using the Power Coordinator to combine the use of the these protocols, more energy can be saved than by using either one of them individually.

## 5. CONCLUSION

In this paper, we have presented link layer support towards realizing a unified radio power management architecture for use in wireless sensor networks. The architectural support we have presented is comprised of two key components: (1) a set of standard interfaces allowing different duty cycling protocols to be implemented on top of a common radio stack, and (2) an abstraction layer that can incorporate the use of different coordination polices for coordinating the duty cycling requirements of multiple applications running on the system.

We have demonstrated the flexibility of this architecture through the development of several different duty cycling protocols, and have conducted experiments in order to evaluate their performance. We have also shown how the architecture can be used to coordinate the duty cycling requirements of multiple applications. By replacing only the coordination policy, we demonstrate that it is possible to coordinate these requirements with those of a backbone based power management protocol as well.

In the future, we plan to build on the ideas presented in this paper to support power management protocols existing at layers other than the link layer. Specifically, we plan to add support for transmission power control protocols existing at the network layer. Another important direction for future work is to integrate this architecture with an overall sensor network architecture [2][22]. A first step in this direction is to develop interfaces that allow it to coordinate with a link layer abstraction such as SP [15]. Integration with SP will enable the support of more efficient power management techniques through fine-grained interactions with network and MAC-layer protocols. Additionally, we plan to integrate this architecture with the power management techniques used by other hardware components (e.g., microcontrollers, sensors, and flash) on a WSN platform. Such a holistic power management approach will result in maximum energy savings in real world systems.

## 6. ACKNOWLEDGEMENTS

This work is supported by NSF NeTS-NOSS grant #CNS-0627126.

## 7. REFERENCES

[1] G.-S. Ahn, E. Miluzzo, A. T. Campbell, S. G. Hong, and F. Cuomo. Funneling-mac: A localized, sink-oriented mac for boosting fidelity in sensor networks. In *Sensys*, 2006.

[2] U. Berkeley. a network architecture for wireless sensor networks. http://webs.cs.berkeley.edu/SNA/.

[3] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-mac: A short preamble mac protocol for duty-cycled wireless sensor networks. In *Sensys*, 2006.

[4] A. Cerpa and D. Estrin. Ascent: Adaptive self-configuring sensor networks topologies. In *INFOCOM*, 2002.

[5] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *MobiCom*, 2001.

[6] Chipcon. Cc2420 radio data sheet. 2004.

[7] A. El-Hoiyi, J.-D. Decotignie, and J. Hernandez. Wisemac: an ultra low power mac protocol for the downlink of infrastructure wireless sensor networks. *Computer Communications*, 1:244– 251, 2004.

[8] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 147–163, New York, NY, USA, 2002. ACM Press.

[9] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149, New York, NY, USA, 2003. ACM Press.

[10] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Trans. Sen. Netw.*, 2(1):1–38, 2006.

[11] IEEE. Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans). In *IEEE Standard 15.4*, 2003.

[12] K. Klues, G. Xing, and C. Lu. Towards a unified radio power management architecture for wireless sensor networks. In *Proceedings of the First International Workshop on Wireless Sensor Network Architecture (WWSNA'07)*, Cambridge, MA, April 25-27 2007.

[13] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, New York, NY, USA, 2004. ACM Press.

[14] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *SenSys*, 2004.

[15] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A unifying link abstraction for wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 76–89, New York, NY, USA, 2005. ACM Press.

[16] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves. Energy-efficient collision-free medium access control for wireless sensor networks. In *SenSys*, 2003.

[17] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-mac: a hybrid mac for wireless sensor networks. In *SenSys*, 2005.

[18] I. Rhee, A. Warrier, J. Min, and L. Xu. Drand: distributed randomized tdma scheduling for wireless ad-hoc networks. In *MobiHoc*, 2006.

[19] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2), 2005.

[20] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *SenSys*, 2004.

[21] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *SenSys*, 2005.

[22] USC. A architecture for tiered wireless sensor networks. http://enl.usc.edu/projects/tenet/.

[23] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys*, 2003.

[24] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *MobiCom*, 2001.

[25] F. Ye, G. Zhong, S. Lu, and L. Zhang. Peas: A robust energy conserving protocol for long-lived sensor networks. In *ICDCS*, 2003.

[26] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated, adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, June 2004.

[27] W. Ye, F. Silva, and J. Heidemann. Ultra-low duty cycle mac with scheduled channel polling. In *SenSys*, 2006.