Report Number: WUCSE-2006-6

2006-01-01

A Unified Architecture for Flexible Radio Power Management in Wireless Sensor Networks

Kevin Klues, Guoliang Xing, and Chenyang Lu

A challenge for many wireless sensor networks is to remain operational for long periods of time on a very limited power supply. While many power management protocols have been proposed, a solution does not yet exist that allows them to be seamlessly integrated into the existing systems. In this paper we study the architectural support required to resolve this issue. We propose a framework that separates sleep scheduling from the basic MAC layer functionality and provide a set of unified interfaces between them. This framework enables different sleep scheduling policies to be easily implemented on top of multiple MAC... **Read complete abstract on page 2.**

A Unified Architecture for Flexible Radio Power Management in Wireless Sensor Networks

Kevin Klues, Guoliang Xing, and Chenyang Lu

Complete Abstract:

A challenge for many wireless sensor networks is to remain operational for long periods of time on a very limited power supply. While many power management protocols have been proposed, a solution does not yet exist that allows them to be seamlessly integrated into the existing systems. In this paper we study the architectural support required to resolve this issue. We propose a framework that separates sleep scheduling from the basic MAC layer functionality and provide a set of unified interfaces between them. This framework enables different sleep scheduling policies to be easily implemented on top of multiple MAC layers. Such a flexibility allows applications to choose the best sleep scheduling policy based on their own particular needs. We demonstrate the practicality of our approach by implementing this framework on top of both the mica2 and telosb radio stacks in TinyOS 2.0. Our micro-benchmark results show that at the cost of a slight increase in code size, our framework significantly eases the development of new radio power management protocols across multiple WSN platforms

2006-6

A Unified Architecture for Flexible Radio Power Management in Wireless Sensor Networks

Authors: Kevin Klues, Guoliang Xing, Chenyang Lu

Corresponding Author: klueska@cs.wustl.edu

Abstract: A challenge for many wireless sensor networks is to remain operational for long periods of time on a very limited power supply. While many power management protocols have been proposed, a solution does not yet exist that allows them to be seamlessly integrated into the existing systems. In this paper we study the architectural support required to resolve this issue. We propose a framework that separates sleep scheduling from the basic MAC layer functionality and provide a set of unified interfaces between them. This framework enables different sleep scheduling policies to be easily implemented on top of multiple MAC layers. Such a flexibility allows applications to choose the best sleep scheduling policy based on their own particular needs. We demonstrate the practicality of our approach by implementing this framework on top of both the mica2 and telosb radio stacks in TinyOS 2.0. Our micro-benchmark results show that at the cost of a slight increase in code size, our framework significantly eases the development of new radio power management protocols across multiple WSN platforms.

Type of Report: Other

A Unified Architecture for Flexible 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}@cs.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 literature, their usage in real-world systems has been limited by the lack of system support for the flexible integration of different power management policies with a diverse set of applications and network platforms. This paper presents the *Unified Power Management Architecture (UPMA)* for supporting radio power management in wireless sensor networks. In contrast to the monolithic approach adopted by existing power management solutions, UPMA provides (1) a set of standard interfaces that allow different sleep scheduling policies to be easily implemented on top of various MAC protocols at the data link layer, and (2) an architectural framework for composing multiple power management policies into a coherent strategy based on application needs. We have implemented UPMA on top of both the Mica2 and Telosb radio stacks in TinyOS-2.0. Microbenchmark results demonstrate that UPMA does not incur a significant decrease in performance when compared to existing monolithic implementations. We also provide a set of case studies that not only demonstrate the flexibility of UPMA, but also its ease of use.

1 Introduction

Energy is a scarce resource in many wireless sensor networks (WSNs). As wireless communication is often a major source of energy consumption, a multitude of radio power management protocols have been developed. Despite fruitful research on various power management protocols, however, it remains difficult for developers to incorporate suitable power management strategies into their systems based on the needs of a specific application. In particular, a key limitation of existing solutions is the lack of system support for *flexible* and *composable* power management strategies for diverse applications and platforms.

Flexibility. Different power management strategies are usually designed for different types of applications. For example, some sleep scheduling protocols [1] save significant energy while introducing considerable communication delays. In contrast, several other protocols [2, 3] are specifically designed to minimize the impact of sleep schedules on communication delays. Moreover, some protocols [4, 5] are optimized for a special class of applications such as data collection that require periodic network traffic. Consequently, a power management protocol that is effective for habitat monitoring applications may be unacceptable for surveillance applications with stringent real-time requirements. It is therefore desirable to allow developers to flexibly select and integrate the most suitable power management protocols into their system based on the specific characteristics of their applications. However, the current power management protocols are often tightly coupled with other system components. As a result, a system is often limited to a specific power management strategy that cannot be easily extended or replaced. For example, commonly used MAC protocols often adopt specific sleep scheduling policies. S-MAC [2] implements a synchronous sleep scheduling algorithm while B-MAC [6], by default, only supports asynchronous sleep scheduling. The implementations of the sleep scheduling policies that exist for S-MAC and B-MAC are highly dependent on the implementations of other MAC level functionality (e.g., CSMA, clear channel assessment, etc.). This dependence restricts applications from being able to choose the best sleep scheduling policy possible, independent from the MAC level functionality they require.

Composability. While earlier research usually focused on developing individual power management protocols in isolation, the presence of multiple tasks running on the same network may require conflicting or complementary power management policies in order to be most efficient. For example, different data collection tasks may specify different duty cycles that result in incompatible sleep schedules. On the other hand, both clustering and sleep scheduling may be used simultaneously to conserve energy while achieving the required level of agility needed by surveillance applications. An appropriate combination of the two strategies may result in further energy savings that cannot be achieved by either strategy alone. Therefore a key challenge faced by devel-

opers is to effectively integrate multiple power management policies into a coherent strategy for a network.

To address the above challenges, we have developed the *Unified Power Management Architecture (UPMA)* to support flexible radio power management in WSNs. In contrast to the monolithic approaches adopted by existing power management solutions, our architecture separates power management strategies from other system functions. Making this separation allows for the effective coordination of multiple power management strategies through a standard set of interfaces and abstractions.

Specifically, we make the following primary contributions in this paper: (1) We design and implement a set of interfaces that allow different sleep scheduling policies to be easily implemented on top of various MAC protocols at the data link layer. This separation gives different applications the ability to choose the sleep scheduling policy that is best suited to its needs. (2) We propose an architectural framework that allows multiple sleep scheduling policies to coexist in a system without interfering with one another. These policies can be implemented independently and then integrated within the architecture as desired. (3) We demonstrate the practicality of our approach by implementing our architecture on top of both the Mica2 and Telosb radio stacks in TinyOS-2.0 [7], the second generation of the TinyOS operating system. (4) We provide micro-benchmark results demonstrating that the architecture does not cause a significant decrease in performance when compared to existing monolithic implementations of the same radio power management strategies. (5) Finally, we provide two case studies that not only demonstrate the flexibility of this architecture, but also its ease of use.

The rest of this paper is organized as follows. Section 2 reviews related work on sensor network architectures and power management protocols. Section 3 presents the design of the UPMA architecture. Section 4 describes the implementation of UPMA on TinyOS-2.0. Section 5 presents the experimental results. Finally, Section 7 concludes the paper.

2 Related Work

Our architecture is related to the on-going research effort in defining sensor network architectures [8, 9]. In particular our work is inspired by and aims to complement the Sensor Network Architecture (SNA) proposed by UC Berkeley. The primary goal of SNA is to create an all encompassing architecture for wireless sensor networks. It aims to support increasingly diverse protocols as well as multiple hardware platforms. Initial steps toward the realization of SNA have been made with the definition of a narrow waist around which technologies at different network layers can be developed independently. This narrow waist is known as the Sensor-net Protocol (SP) [10], and exists between the network and data link layers of a traditional networking protocol stack. SP achieves flexibility by allowing multiple data link and network technologies to be used simultaneously on a single node, while preserving efficiency by allowing them all to share data in a standardized way. Although various power management strategies can be made to run both above and below the SP layer as desired, there is no facility built into SP that allows for the flexible integration of multiple radio power management strategies interacting across multiple layers. To our knowledge, UPMA is the first unified architecture for flexible radio power management in WSNs.

Existing approaches to radio power management fall into two basic categories: transmission power control and sleep scheduling. Transmission power control [11] reduces the energy consumed for transmission by adjusting the transmission power of the radio. Sleep scheduling reduces energy wasted during idle listening by scheduling the radio to sleep. The architecture presented in this paper is only designed for use with sleep scheduling protocols. The architectural support for transmission power control is left as future work.

Sleep scheduling has proven to be very efficient and can extend the system lifetime of WSNs by many orders of magnitude. Two basic types of sleep scheduling protocols exist: synchronous and asynchronous. In synchronous protocols, each node in a network runs with the same (or similar) duty cycle as its neighbors, and they are all synchronized to begin their wake-up/sleep intervals at the same time. Several energy-efficient MAC protocols that use synchronous sleep scheduling include S-MAC [2], IEEE 802.15.4 [12] and IEEE 802.11 PSM [13]. In contrast to synchronous sleep scheduling protocols, asynchronous ones allow individual nodes to wake up and go to sleep on their own schedules[1]. Asynchronous sleep scheduling can be found in the various TDMA family of protocols [14] and recent delay-efficient sleep protocols [15, 6, 16, 17].

Each type of power management protocol has its own pros and cons. For instance, asynchronous protocols, such as the Low Power Listening (LPL) scheme implemented in B-MAC, do not require clock synchronization among nodes and can adapt to changing network activity. They may introduce additional energy costs, however, by transmitting long preambles and unnecessarily waking up nodes due to overhearing. Synchronous protocols, on the other hand, may be particularly energy efficient when used with applications that have predictable workloads. While these protocols do require the overhead of clock synchronization, they can be designed to wake up nodes "just in time", based on the timing patterns of applications in the network. One limitation of synchronous protocols, however, is their long delay in communication due to a long period of inactivity when all nodes are synchronized to sleep. Recently, several adaptive sleep techniques [2, 3, 5] have been proposed that help to mitigate the impact of sleep scheduling on communication delay. Instead of proposing yet another sleep scheduling protocol, our work focuses on developing a unified architecture for flexibly integrating the use of different radio management protocols within a single WSN system.

3 Design of UPMA

Supporting a diverse set of power management strategies in wireless sensor networks requires a strong architectural foundation. The architecture itself must be powerful enough to support a wide range of existing power management solutions, yet flexible enough to scale as more innovative tech-

[1]Nodes using these protocols may still require clock synchronization to determine the time slot in which they should operate

nologies continue to emerge. Specifically, this architecture must be able to meet the following criteria. (1) It must be scalable, i.e. the set of interfaces defined must be rich enough to support all existing strategies as well as strategies that have not yet been proposed. (2) It must allow for composability, i.e. all power management strategies built within this architecture must adhere to the use of the same set of interfaces for interacting with the rest of the system. (3) Individual power management strategies should have independent implementations, and network protocols/applications should not rely on the presence of any particular one. (4) It should be possible to spread each strategy across multiple layers in the traditional network protocol stack, and any number of individual strategies should be able to coexist at each layer. (5) It should support the creation of cross-layer coordination polices that govern the interaction between power management strategies existing at different layers.

In this section we present a radio power management architecture that meets each of the requirements specified above. While our discussion is limited to the support of sleep scheduling polices alone, the ideas presented can be expanded to support power management strategies that exist across multiple layers.

3.1 Architectural Overview

Power management strategies require that applications specify some of the parameters they need in order to operate. For example, S-MAC requires an application to specify a radio's duty cycle period, while B-MAC requires the time between different check intervals to be specified. If a single power management strategy is used by multiple applications on the same node at the same time, each of these applications may specify different, or even conflicting values for some of these parameters. A method is needed to combine the parameters supplied by each application into a single coherent set of data. Once these parameter values have been aggregated, the result can be passed on to the radio power management strategy in use. Since some strategies may be more appropriate for different application scenarios than others, and the choice of which strategy to use should not depend on the specific type of radio in use, a set of interfaces must be defined that allow each power management strategy to control the power state of the radio in a standardized way. These interfaces must be specific enough that they encapsulate all of the functionality required by each of the radio power management protocols existing today, yet generic enough that they are usable by power management strategies that will be developed in the future.

When the only power management strategies existing in a system consist of sleep scheduling protocols, the architectural support required to satisfy the 5 criteria outlined in section 3 can be seen in Fig. 1.

The following subsections describe each of the components depicted in Fig. 1 in greater detail. The *Power Management Abstraction* component is described first, followed by a discussion of the the *Power Manager* component along with its internal *Aggregator*. This section concludes with a description of the set of interfaces that need to be exposed by each radio implementation in order to allow various sleep scheduling protocols to be developed on top of them.

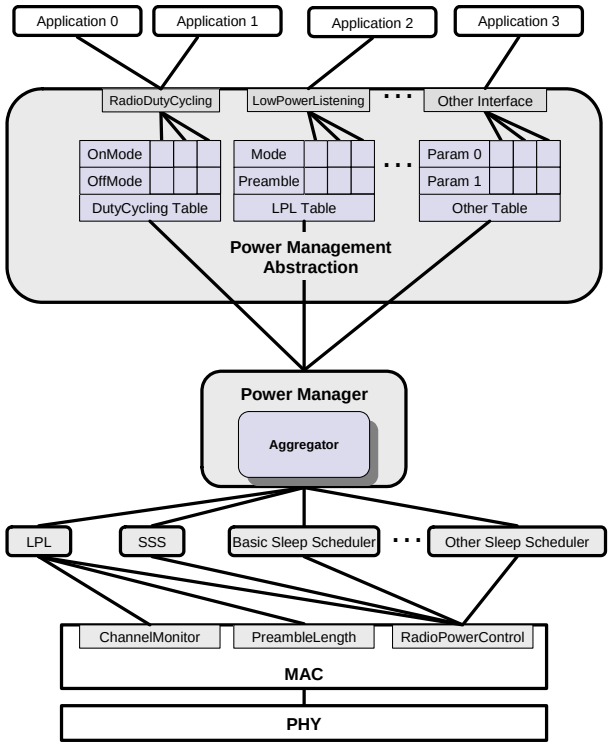


Figure 1. Radio Power Management Architecture

3.2 Abstraction Component

The *Power Management Abstraction* is functionally equivalent to a set of buffers used to store the parameters supplied to a sleep scheduling policy by each of its users[2]. These buffers take the form of a table, with one table existing per interface. The rows in each table are used to store each of the parameters supplied through that tables interface, while the columns are used to separate those parameters specified by different applications. By keeping the parameters required by each interface in a separate table, only those tables mapping to the actual power management strategy in use will need to be included at any given time. By organizing these tables in such a way, the amount of memory required by the application is significantly reduced. Furthermore, by allowing an application to provide parameters to each interface as if it were communicating directly with the power management policy itself, applications can be developed without consideration for other applications that could potentially coexist along with them.

Figure 1 shows how this abstraction layer can be used to store parameters supplied through the `RadioDutyCycling` and `LowPowerListening` interfaces. These interfaces will be described in greater detail in section 4.1. As newer sleep scheduling polices are developed, each with their own set of newly defined interfaces, tables can be created for use in this abstraction layer as needed. In this way, the architecture is scalable as required by criterion (1) from Section 3.

[2]These users could be applications, packet schedulers, etc. Although Fig. 1 shows each of them to be an application, they could potentially exist at any level of the network protocol stack

3.3 Power Manager Component

The *Power Manager* component is used to perform two separate functions. It is used to both aggregate parameters supplied to each power management strategy through the *Power Management Abstraction* as well as coordinate the use of multiple power management strategies so that they do not interfere with one another.

In the same way that the *Power Management Abstraction* allows multiple applications to coexist without knowledge of one another, the *Power Manager* allows multiple power management strategies to coexist. They can be implemented as if they are the only strategy that will ever exist in any given system at any given time. The combination of these two architectural components provides an intermediate layer between applications and power management strategies that allow each of them to be developed as if they communicate directly with one another. The burden of coordinating their usage is left to the *Power Manager* component. A developer may easily change the coordination approach by replacing it with the appropriate aggregation policy. In this way, criteria (3), (4), and (5) from section 3 have been met. Namely, that each power management strategies has its own implementation, independent of the network protocols that use them (3), any number power management strategies can be used in the system simultaneously (4), and coordination between different power management polices is possible using the aggregation component (5). Two example aggregation policies are described in Section 4.

3.4 Interfaces with the MAC Layer

The final criterion specified in Section 3 that has not yet been met (2) is to allow for composability, forcing all power management strategies built within this architecture to adhere the same set of interfaces for interacting with the rest of the system. In order to meet this criterion, a standard set of interfaces must be defined for allowing different power management procotols to communicate with the radio. As our investigation is limited to sleep scheduling policies alone, we only identify those interfaces required by power management strategies of this type. They can all be exposed through the MAC layer of a traditional radio stack implementation.

Existing MAC protocols can be broken up into three different classes: contention based, non-contention based, and hybrid [18]. Contention based MAC protocols usually perform CSMA and rely on clear channel assessment (CCA) to determine if a radio channel is free or not. Examples of contention based MAC protocols include B-MAC and S-MAC. Non-contention based ones, rely on TDMA schedules to determine when packets should be sent, and do not require any sort of clear channel assessment to be performed. TRAMA [19] is a typical TDMA MAC protocol. Hybrid based protocols such as 802.15.4 [12] and Z-MAC [20] are a mixture of both.

Some sleep scheduling policies rely on the existence of a particular type of MAC while others do not. For example, the sleep scheduling policy used by B-MAC (Low Power Listening) needs to perform CCA in order to determine whether the radio should be active or not. Since only CSMA based MAC protocols provide this sort of functionality, Low Power Listening is compatible with them alone. TDMA based protocols, on the other hand, do not rely on any special interfaces for performing CCA. They only require an interface for turning the radio on and off at intervals based on their time schedules. Hybrid protocols obviously require both. Figure 2 shows the interfaces necessary to support sleep scheduling policies of all types.

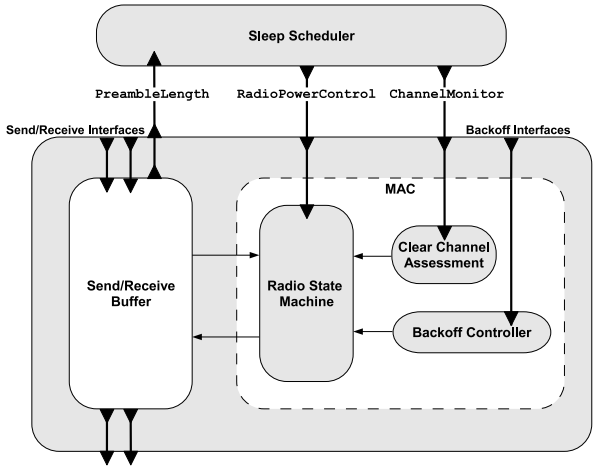


Figure 2. Proposed MAC level architecture with sleep scheduler outside of radio implementation

Three interfaces are made available through the MAC layer for use by different sleep scheduling protocols. Although, each of the interface definitions provided in this section are written in nesC [21], the architectural ideas presented are fundamental, and do not rely on any implementation in particular. If a particular MAC layer is unable to provide one of these interfaces it may be limited to the types of sleep scheduling policies that can be built on top of it. Ideally all radio stack implementations should be able to provide each of these interfaces, but in practice this may not always be possible. Each interface is described in greater detail below.

The `RadioPowerControl` Interface:
The first of these interfaces is the `RadioPowerControl`. This interface allows a radio to be switched between its on and off power states. It must be implemented by all types of MAC protocols, since without it, power control 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 user of this interface calls the `on()` command to put a radio into the "on" state and begin receiving transmissions. Once the radio has been fully switched on, the `onDone()` event is signalled to signify that the operation has completed. The same holds true for the `off()` command and its corresponding `offDone()` event when powering the radio down.

The `ChannelMonitor` Interface:
The second interface is the `ChannelMonitor` interface. This interface is used to expose the clear channel assessment (CCA) capabilites of the radio. This interface may only be implementable by CSMA based MAC protocols. If this interface is not exposed, the use of certain sleep scheduling 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 `check()` command can be called by a sleep scheduling policy to determine if a radio channel is busy or not. If the CCA algorithm implementing this interface determines that the channel is busy, it signals the `busy()` event. If it determines that the channel is free, it signals the `free()` event. If for some reason the CCA algorithm cannot complete its operation, it will signal an `error()` event to allow the sleep scheduler to decide what it should do next.

The `PreambleLength` Interface:
The third and final interface is the `PreambleLength` interface. This interface allows a sleep scheduling policy to dynamically change the length of the preamble associated with a particular outgoing packet. The exposure of this interface through the MAC layer has been motivated by a set of representative asynchronous sleep scheduling protocols such as the Low Power Listening scheme implemented in B-MAC. In this protocol, the number of preambles sent with each packet needs to be set dynamically. The exposure of this interface will most likely play an important role in the development of future sleep scheduling policies based on Low Power listening.

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

4 Implementation

An implementation of the architecture described in the previous section has been created for TinyOS-2.0. We have chosen to use TinyOS-2.0 as our implementation platform since it is still maturing and does not yet have many protocols developed for it. Our hope is that as people start moving implementations of their power management protocols from TinyOS-1.x into TinyOS-2.0 in the near future, they will do so within the architecture presented here.

In this section we first present how we have exposed the appropriate interfaces through the MAC layer implementations in TinyOS-2.0. We provide sample implementations of both Low Power Listening and the Simple Synchronous Sleeping (SSS) on top of these newly exposed interfaces. A third policy called Basic Synchronous Sleeping (BSS) is also introduced that is functionally similar to SSS but provides a different type of interface to the user. By providing implementations of both synchronous, and asynchronous sleep scheduling policies we are able to show the flexibility of the architecture in allowing them to exist on top of two very different radio platforms (Mica2 and Telosb).

An implementation of the entire UPMA is also presented. We have created two different instantiations of the architecture, using two different sets of power management policies and two different aggregation polices. We use the two polices to demonstrate how multiple applications can be made to interact with multiple sleep scheduling protocols. By a simple change of aggregation policy we are able to change the semantics of how this interaction takes place. Implementations of the applications and sleep scheduling policies are never altered, and the only changes made are within the aggregation components themselves.

4.1 Interfaces with the MAC Layer

In this section we show how to use the interfaces between UPMA and the MAC layer to implement both synchronous and asynchronous sleep scheduling policies on top of B-MAC in TinyOS-2.0. In order to do this, the original B-MAC implementation needed to be modified in two distinct ways. First, the built in sleep scheduling policy associated with B-MAC (Low Power Listening (LPL)) had to be removed. Second, our newly presented interfaces had to be exposed through the MAC layer so that they could interact with an external sleep scheduler implementation. Three different sleep scheduling protocols were built on top of B-MAC using these newly exposed these interfaces: LPL, SSS, and BSS. LPL is the traditional asynchronous sleep scheduling policy that was previously built into B-MAC, while SSS and BSS are two synchronous policies that have been developed to demonstrate the generality of the interfaces.

In our current implementation, the modified B-MAC exposes the complete set of interfaces described in section 3.4 on the Mica2 radio stack, while only a subset are exposed on Telosb. Due to the limitations of Chipcon CC2420 radio[3], only the `RadioPowerControl` interface has been exposed on the Telosb radio. Because of this limitation, the new LPL has only been implemented on the Mica2 radio stack, while SSS and BSS have been implemented on both of them. All implementations can be easily ported to other radio platforms supporting the set of standard interfaces specified by UPMA. The following three subsections describe the implementation of these sleep scheduling protocols in more detail.

4.1.1 Low Power Listening

Low power listening allows a radio to sleep for long periods, waking up periodically to check if a packet is coming

[3]In the CC2420 radio, the clear channel assessment algorithm is internal to the radio chip and not easily exposed to external components as part of the `ChannelMonitor` interface. The CC2420 radio hardware also limits the packet size, including any preamble bytes necessary. The `PreambleLength` interface would therefore not allow preambles of arbitrary length. Note that the same limitations also make it difficult to implement LPL even if it were inside B-MAC

in on the radio channel or not. If no packet is present, it goes back to sleep until the next time it is supposed to check. Packets are sent with preamble lengths equal to the size of each node's sleep interval so that no packets will be dropped simply because its destination node was asleep when it was sent. It 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. Figure 3 shows our new implementation of LPL as a separate component, and how it uses the standard interfaces defined in the previous section to interact with the modified B-MAC that no longer includes LPL.

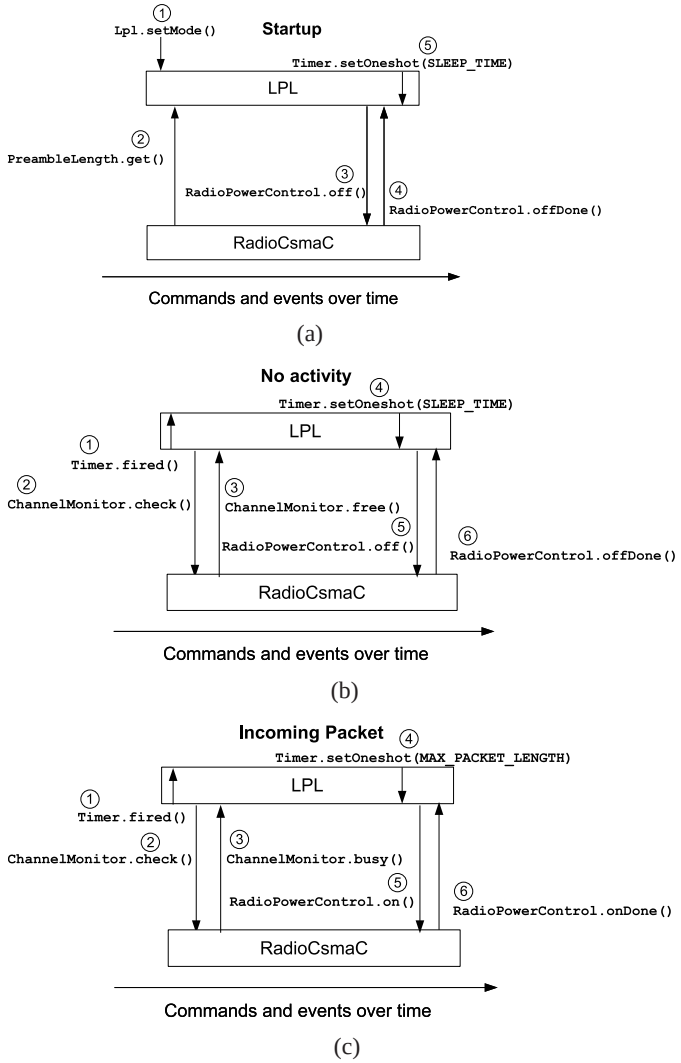


Figure 3. Interaction of platform independent LPL implementation with radio level interfaces

This figure depicts three situations in which the new LPL may use the interfaces provided by the MAC layer to perform its sleep scheduling duties. During startup (Figure 3(a), an application specifies the LPL mode of operation (i.e. the check interval to use along with its corresponding preamble length) through the `Lpl` interface provided by LPL (1). The MAC layer then retrieves the newly specified preamble length through the `PreambeLength` interface (2), and LPL turns the radio off through the `RadioPowerControl` interface (3). Once the radio has been completely shut down (4), a timer is set based on the check interval specified by the application (SLEEP_TIME)(5).

One of two conditions could then occur. In both cases (Figure 3(b), 3(c)) the timer will expire (1) and the channel will be checked for activity (2). An event will then come back signifying that the channel is either busy or free (3), and the timer will be reset with one of two values. If the channel was free (Figure 3(b)), the timer is reset to its check interval length (SLEEP_TIME)(4) and the radio is shut off (5)(6). If the channel is busy, however,(Figure 3(c)) the the timer is set to allow an entire packet to be received (MAX_PACKET_LENGTH)(4), and the radio is turned fully on (5)(6).

4.1.2 Simple Synchronous Sleeping

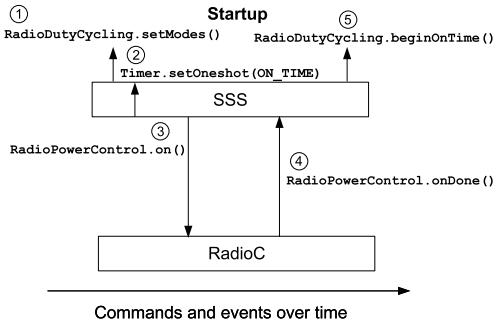
We have designed and implemented a synchronous sleep scheduling protocol known as SSS (Simple Synchronous Sleeping) on top of the separated version of B-MAC. SSS relies on time synchronization of all nodes in a network to precisely control their duty cycles. The duty cycle of the radio is tunable through the following interface.

```
interface RadioDutyCycling {
  command error_t setModes(uint8_t onMode, uint8_t offMode);
  command error_t setOnTimeMode(uint8_t onMode);
  command error_t setOffTimeMode(uint8_t offMode);
  event void beginOnTime();
  event void beginOffTime();
}
```

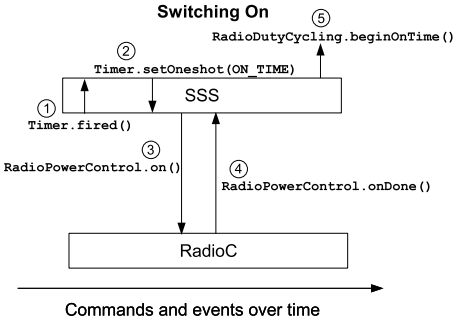
A higher layer uses this interface to set the duty cycle of the radio and be notified whenever it has been switched on or off. Since the start of every radio's duty cycle must be synchronized, all nodes having the same duty cycle will be able to communicate with each other during the on time of the radio and conserve energy during the off time. Figure 4 shows our implementation of SSS and how it uses the interfaces presented in the previous section to interact with the radio.

During startup (Figure 4(a)), an application specifies the mode of operation for SSS (length of on and off times within a single duty cycle period) through the `RadioDutyCycling` interface (1). A timer is then set to the on time specified by the application (2), and the radio is switched on (3). Once the radio has been fully switched on (4), the application is signaled notifying it of this event (5). After this timer expires, SSS alternates the on and off states of the radio according to the time intervals specified by the application. The steps taken in each situation can be seen in in Figure 4(a) and Figure 4(b) respectively.

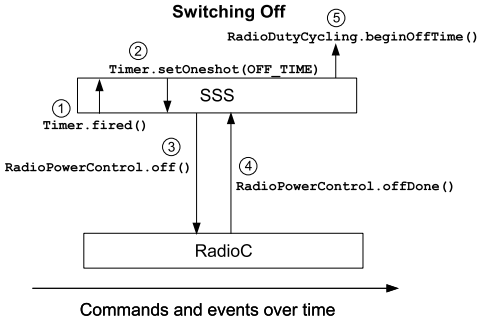
Our implementation of SSS shows that a synchronous sleep scheduling protocol can be built on top of the standard interfaces exposed by our MAC layer implementation. It is conceivable that MAC protocols such as S-MAC that have synchronous sleep scheduling protocols built into them should be separable in the same way that LPL was separable from B-MAC. In the future, it is hoped that making such sep-



(a)



(b)



(c)

Figure 4. Interaction of platform independent SSS implementation with radio level interfaces

arations will be unnecessary and that MAC protocols will be implemented without sleep scheduling protocols in the first place. They will instead be implemented on top of the MAC through the interfaces presented here.

4.1.3 Basic Synchronous Scheduler

This general purpose sleep scheduler can be told by an application directly when (as well as for how long) to power the radio on and off using the following interface.

```
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. BSS will signal the `ready()` event to a user whenever it is ready to turn the radio either on or off. Once this event returns, the radio will be either turned on or off based on whether `turnOnFor()` and `turnOffFor()`, was called most recently. A timer will be started based on the value supplied to that command. In order to duty cycle a radio, the user of the `DutyCycleTimes` interface can alternate calls to `turnOnFor()` and `turnOffFor()` commands within the body of the `ready()` event. If no calls to `turnOnFor()` or `turnOffFor()` are made between subsequent `ready()` events, the timer is restarted with the same value it had previously, and the power state of the radio remains unchanged.

BSS is similar to SSS in many ways. Both SSS and BSS require time synchronization for all nodes in a network. They both turn the radio on and off for certain time durations as specified by the user. They also both interact with the MAC layer through the `RadioPowerControl` interface. The primary difference between the two is that SSS allows an application to specify a periodic radio duty cycle, while BSS requires that the on and off time be specified each time the radio is ready to make another transition. While BSS may be more general, SSS can be much more convenient for certain applications.

4.2 Duty Cycle Aggregation

We have implemented an instantiation of UPMA that uses the aggregation component to combine the duty cycling requirements of multiple applications. This implementation of UPMA allows applications running simultaneously to independently specify their own duty cycling requirements through the `RadioDutyCycling` interface of UPMA. The Power Management Abstraction keeps track of the on and off times supplied by each application, and the Power Manager aggregates this data to produce a sleep schedule that combines the requirements of all applications into a single coherent schedule. The Power Manager then uses this schedule to inform BSS of the next on and off intervals through its `DutyCycleTime` interface, and BSS turns the radio on and off accordingly.

Specifically, the Power Manager aggregates the duty cycles of multiple applications according to an OR policy as shown in Figure 5.

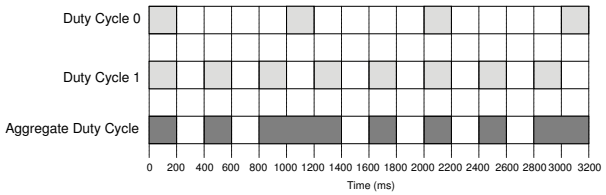


Figure 5. Aggregation of multiple duty cycles

This aggregation policy is implemented as follows. (1) All duty cycles are shifted to begin at the same time instant. (2) They all run periodically according to their own schedule. (3) If *any one* 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 schedule of on and off times resulting from this policy may not be expressible using the original `RadioDutyCycling interface`. It will be periodic in nature, but may contain multiple on and off durations within each of its cycles. In Figure 5 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 the periods of each duty cycle being aggregated needs to be determined. In this case it is 2000ms. Since multiple on and off periods will exist within this period, BSS is the most appropriate choice for executing the aggregated schedule.

4.3 Duty Cycle/PEAS Aggregation

The second instantiation of UPMA we have developed demonstrates how to coordinate the use of two different power management strategies with multiple applications running on top of them. To present this policy, we introduce an existing power management protocol known as PEAS [22]. PEAS is a backbone maintenance protocol for wireless sensor networks that can be used to control the density of active nodes in a network as well as the frequency with which new nodes will become active once those active nodes start to die out. 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. Nodes that become active begin running a set of applications as well as send PEAS reply messages in the background as appropriate. Inactive nodes do not start their applications and simply continue to send probing messages at a very low duty cycle.

An aggregation policy has been created that allows the functionality provided by PEAS to be coordinated with the duty cycles specified by applications through the `RadioDutyCycling` interface. For nodes that PEAS has declared inactive, only the PEAS duty cycle is allowed to run, and all applications become disabled. For nodes that PEAS designates as active, the duty cycles of all applications are aggregated together according to the OR policy described in the previous section. Active nodes also continue to run the PEAS duty cycle in order to be able to reply to the probing messages sent by any inactive nodes.

By controlling the spatial density of active nodes in a network, PEAS is able to provide spatial energy savings. Duty cycling a node, on the other hand, provides temporal energy savings by dividing up the time a node is either active or inactive within a given time period. By combining the energy benefits provided by PEAS with those of duty cycling the radio, the aggregation policy described in this section allows more energy to be saved in a network than the use of either one individually. A key advantage of UPMA is that neither the implementation of PEAS nor the implementation of any application needs to be altered in order to achieve these energy savings.

5 Evaluation

The previous section described sample implementations of each of the key components required by UPMA. This section provides experimental results exploiting the use of these implementations.

The first part of this section provides experimental results showing the plausibility of separating B-MAC from its built in Low Power Listening policy and exposing the `RadioPowerControl`, `ChannelMonitor`, and `PreambleLength` interfaces. We provide results comparing the original B-MAC implementation on Mica2 with our newly separated one, as well as evaluate the effectiveness of using SSS on top of both the Mica2 and Telosb MAC layer implementations.

The second part of this section provides results for evaluating the two different instantiations of UMPA described in the previous section. The first experiment demonstrates that this architecture has the ability to combine multiple duty cycles in a way that is transparent to each of the applications specifying those duty cycles. The second experiment shows how multiple sleep scheduling policies can be combined together to achieve greater energy savings than each of them could achieve individually.

5.1 Low Power Listening

The first set of experiments involve comparing the original B-MAC implementation for the CC1000 radio on mica2 to our new implementation of B-MAC that includes LPL as a separate component. Our experimental settings are the same as the ones presented in [6]. We also compare the difference in the code size between the two implementations. By showing that our implementation of B-MAC 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 good a framework for B-MAC to be implemented in as the original one. Since our framework does not limit one to using LPL implementation as its default sleep scheduling policy, however, it provides much more flexibility. We performed the following sets of experiments.

Throughput vs. Number of Nodes:
There is one receiver, with a variable number of senders from 1 to 4 all 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 (kbits per second) at the receiver over 2 minutes.

Latency vs. Number of Hops:
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.

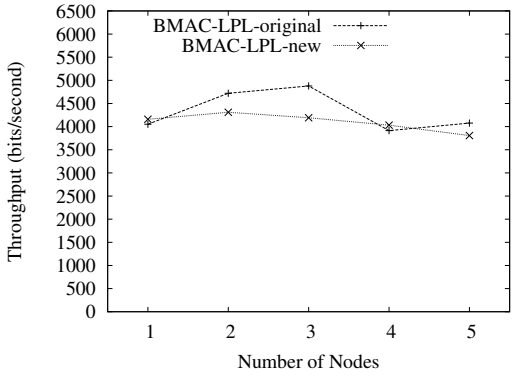


Figure 6. Throughput vs. Number of nodes at 100% duty cycle for the two different LPL-BMAC implementations

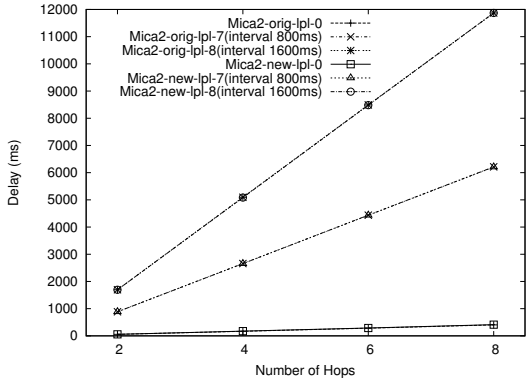


Figure 7. Latency vs. Number of Hops at different LPL check intervals for the two different LPL-BMAC implementations

Figures 6 and 7 show exactly how similar the original LPL implementation performs in comparison to the one using our proposed framework. This behavior is in fact expected, since they do indeed implement the exact same protocol. Table 1 shows the difference in compiled code size between the new and old LPL-BMAC implementations for the two different applications used in the above experiments.

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 B-MAC, 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 between the new B-MAC and LPL layers.

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

Table 1. LPL Memory Footprint

5.2 Simple Synchronous Sleeping

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 MAC layer implementations. Results are given for both Mica2 and Telosb.

The setup for each experiment found in this section are exactly the same as those described for LPL in the last section. For measuring throughput vs. number of nodes, SSS was run at duty cycles of 100%, 47%, and 20%, and the total throughput was measured over 2 minutes. For measuring latency vs. number of hops, SSS was ran at a 50% duty cycle, and the average latency from source to sink for a single packet was measured.

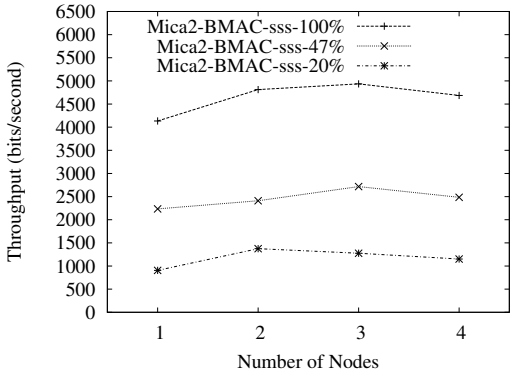


Figure 8. Throughput vs. Number of nodes at different duty cycles for the SSS-BMAC implementation on mica2

Figures 8 and 9 show that SSS is able to deliver more data on both Mica2 and Telosb when the radio duty cycle is higher. It is not surprising that 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 used by Mica2.

Figure 10 demonstrates that SSS is able to synchronize the on time of multiple nodes in a multihop network. If all on times were not synchronized, then some packets would undoubtedly have been dropped between the source and the sink.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.

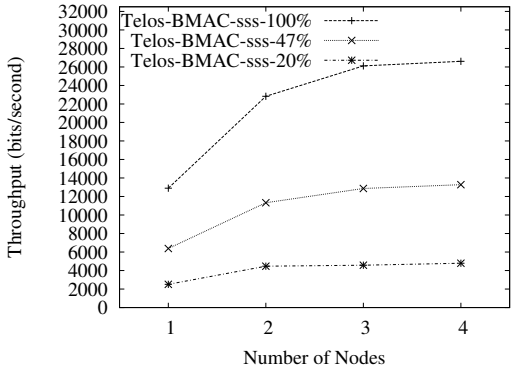


Figure 9. Throughput vs. Number of nodes at different duty cycles for the SSS-BMAC implementation on telosb

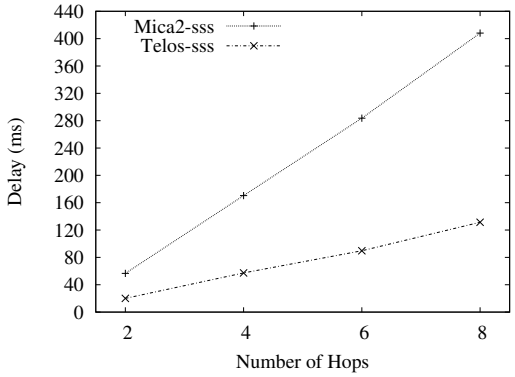


Figure 10. Latency vs. Number of Hops at 50% duty cycle for the SSS-BMAC implementation on both mica2 and telosb

Overall, the results of these experiments have shown the following: (1) Implementing LPL using our framework does not have any adverse affects on its performance. (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 asynchronous and synchronous sleep scheduling policies can be easily implemented on top of these interfaces in both a platform independent and MAC level independent manner.

5.3 Combining Multiple Duty Cycles

In this subsection we evaluate the instantiation of UPMA that combines duty cycles specified by multiple applications. 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 4.2.

This experiment is conducted by systematically increasing the number of applications present in the network at any given time. 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 11 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.

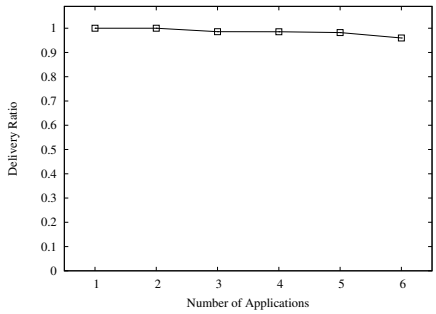


Figure 11. The delivery ratio measured at the master node.

Figure 12 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 12, performing the combination will always yield a lower overall duty cycle than not performing it.

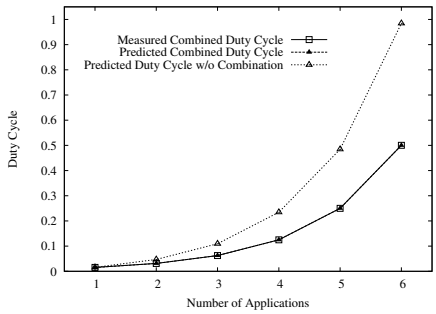


Figure 12. The duty cycle of the master node.

The results in this section demonstrate that UPMA 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.

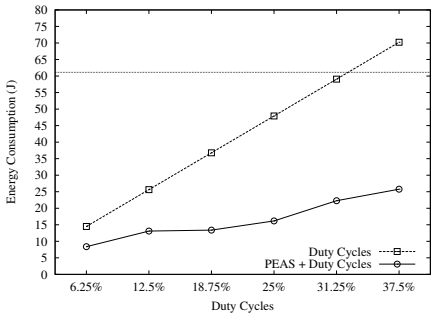


Figure 13. The total energy consumption of the network.

5.4 Combining Duty Cycles with PEAS

In this section we evaluate the instantiation of UPMA that combines PEAS with applications that are able to specify their own duty cycles. The network consists of a master Telosb node and 15 slave Telosb nodes placed within a 5×3 grid. Each slave node runs both PEAS as well as an application that is able to specify its own duty cycle. Six applications are possible, each 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. PEAS runs with a duty cycle period of 16s and an on time of 200ms. Inactive nodes send PEAS probe messages at some random time within their 200ms on period, and active nodes send a packet to the master node at some random time during their on period. Although all nodes are within communication range of each other, 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 [23][4].

Figure 13 shows the total energy consumption of all nodes in the network using three different power management policies. The energy consumed using each policy is measured, and their results are compared. The first policy matches the one described in section 4.3. Under this policy, the radio is turned on and off according to the duty cycle of the sensing application only if PEAS chooses it to be one of its active nodes. All other nodes only wake up and perform the probing process of PEAS every 16s and remain asleep at other times. In the second policy, all nodes operate according to the duty cycle specified by their applications, and PEAS functionality is disabled. In the third policy, the opposite is true. All nodes only run PEAS, and the duty cycling capabilities of each application are disabled. As a baseline for comparison, the energy consumed using this third policy is shown in Figure 13 by a level straight line.

We can see from Figure 13 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 implemented in UPMA 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 UPMA framework. By using the Power Manager component to combine the sue of the these protocols, more energy can be saved than by using either one of them individually.

6 Conclusion

We have presented UPMA, a unified architecture for flexible radio power management in wireless sensor networks. UPMA is comprised of three key components: (1) a power management abstraction that allows multiple applications and protocols to specify their desired sleep policies independently; (2) a power manager that aggregates multiple policies into coherent sleep schedules; and (3) a set of standard interfaces allowing sleep scheduling policies to be separated from MAC layer implementations, thus enabling different combinations of sleep scheduling protocols and MAC protocols. We have demonstrated the flexibility of UPMA through two case studies in which different sets of sleep scheduling policies have been incorporated into this architecture using two different aggregation policies. We have also demonstrated that the separation of sleep scheduling from the data link layer only introduces minimum overhead on Mica2 and Telosb radio stacks on TinyOS-2.0.

In the future we plan to further explore and enhance the flexibility of UPMA by developing new approaches for aggregating different power management protocols. For instance, a simple but conservative approach for aggregating multiple asynchronous sleeping policies (e.g., LPL) is to use the minimum check interval and the longest preamble of all active applications. Other aggregation policies may achieve more energy savings than this simple approach. Furthermore, a more sophisticated form of aggregation might involve some sort of cross-layer optimization (e.g., sleep scheduling and power-aware routing). A goal of UPMA is to support complex cross-layer optimization policies to be implemented in the aggregator, without requiring modifications to any other system components.

An important direction for future work is to integrate UPMA within an overall sensor network architecture. A first step in this direction is to develop interfaces that allow

[4]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.

UPMA to coordinate with a link layer abstraction such as SP[5]. The integration with SP may potentially enable UPMA to support more efficient power management techniques through fine-grained interactions with network and MAC-layer protocols. Another promising area of future work is to integrate UPMA with the power management of other hardware components (e.g., microcontrollers, sensors and flash storage controllers) on a WSN platform. Such a holistic power management approach will result in maximum energy savings in real world systems.

7 References

[1] T. van Dam and K. Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *Sensys*, 2003.

[2] 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.

[3] R. Zheng and R. Kravets, "On-demand power management for ad hoc networks," in *INFOCOM*, 2003.

[4] N. Ramanathan, M. Yarvis, J. Chhabra, N. Kushalnagar, L. Krishnamurthy, and D. Estrin, "A stream-oriented power management protocol for low duty cycle sensor network applications," in *In Proceedings of the Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, 2005.

[5] O. Chipara, C. Lu, and G.-C. Roman, "Efficient power management based on application timing semantics for wireless sensor networks," in *International Conference on Distributed Computing Systems (ICDCS)*, 2000.

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

[7] P. Levis, D. Gay, V. Handziski, J.-H.Hauer, B.Greenstein, M.Turon, J.Hui, K.Klues, C.Sharp, R.Szewczyk, J.Polastre, P.Buonadonna, L.Nachman, G.Tolle, D.Culler, and A.Wolisz, "T2: A second generation os for embedded sensor networks," Telecommunication Networks Group, Technische Universität Berlin, Tech. Rep. TKN-05-007, Nov. 2005.

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

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

[10] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica, "A unifying link abstraction for wireless sensor networks," in *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys)*, 2005.

[11] P. Santi, "Topology control in wireless ad hoc and sensor networks," *ACM Comput. Surv.*, vol. 37, no. 2, pp. 164–194, 2005.

[12] 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.

[13] ——, "Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Standard 802.11*, 1999.

[14] K. Oikonomou, N. B. Pronios, and I. Stavrakakis, "Performance analysis of topology-unaware tdma mac schemes for ad hoc networks with topology control." *Computer Communications*, vol. 28, no. 3, pp. 313–324, 2005.

[15] G. Lu, N. Sadagopan, B. Krishnamachari, and A. Goel, "Delay efficient sleep scheduling in wireless sensor networks," in *IEEE INFOCOM*, 2005.

[16] Q. Cao, T. F. Abdelzaher, T. He, and J. A. Stankovic, "Towards optimal sleep scheduling in sensor networks for rare-event detection," in *IPSN*, 2005.

[17] R. Zheng, J. C. Hou, and L. Sha, "Asynchronous wakeup for ad hoc networks," in *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM Press, 2003, pp. 35–45.

[18] J. A. Stankovic, T. Abdelzaher, C. Lu, L. Sha, and J. Hou, "Real-time communication and coordination in embedded sensor networks," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1002–1022, 2003.

[19] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves, "Energy-efficient collision-free medium access control for wireless sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003.

[20] I. Rhee, A. Warrier, M. Aia, and J. Min, "Z-mac: a hybrid mac for wireless sensor networks," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, 2005.

[21] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to network embedded systems," in *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, 2003.

[22] F. Ye, G. Zhong, S. Lu, and L. Zhang, "Peas: A robust energy conserving protocol for long-lived sensor networks," in *The 23rd International Conference on Distributed Computing Systems (ICDCS'03)*, May 2003, pp. 169–177.

[23] Chipcon, "Cc2420 radio data sheet," 2004.

[5] We have not implemented the interfaces with SP because it is not available on TinyOS-2.0 and has only recently been made available in the Boomerang version of TinyOS-1.x.