An Architecture for Energy Management in Wireless Sensor Networks

Xiaofan Jiang[†], Jay Taneja[†], Jorge Ortiz[†], Arsalan Tavakoli[†], Prabal Dutta[†], Jaein Jeong[†], David Culler[†⋆], Philip Levis[‡], and Scott Shenker[†]

[†]UC Berkeley EECS Dept. Berkeley, California 94720
[‡]Stanford CS Dept. Stanford, California 94305
[⋆]Arch Rock Corporation San Francisco, California 94105

1 Introduction

Sensornets are becoming more widely adopted for commercial and scientific use and, in settings where battery replacement or recharging is difficult, it is important that sensornets have long and predictable lifetimes. We thus expect energy management to play an increasingly important role in meeting user requirements. Today, system developers seek a balance between network lifetime and performance, but recent history shows that unexpected and dynamic environmental conditions often scuttle expected energy budgets.

For example, many nodes in the Great Duck Island deployment were conjectured to have died prematurely because unexpected overhearing of traffic caused radios to become operational for longer than originally predicted [22]. This pattern was repeated in the Redwoods deployment, but for a supposedly different reason: some nodes seemingly died prematurely because they became disconnected from the wireless network and depleted their batteries trying to reconnect [24]. Even systems augmented with energy harvesting are still susceptible to this type of problem. In the Trio Testbed, seasonal and daily variations in solar power, the angle of inclination of the solar cell, the effect of dirt and bird droppings on the cells, and the inefficiencies in power storage and transfer resulted in node duty cycles ranging from 20% to 100% [5].

The issues with these deployments arise from mistaken assumptions, unforeseen difficulties, and unpredictable environmental dynamics. Solutions to these issues take two extreme approaches. At one extreme, some have proposed runtime adaptation to meet lifetime requirements [16] or energy availability [11, 10]. While promising, these efforts have addressed rather coarse-grained, high-level adaptation – for example, by adjusting sampling rates or varying the system-wide duty cycle – but they remain silent on prioritizing energy usage in a fine-grained and flexible manner. At the other extreme, low-level energy management mechanisms that give direct control over the hardware to multiple entities (e.g. network protocols) can be tedious to implement and difficult to debug because of the lack of any isolation. Arbitration can address the isolation problem, but it does not enable runtime adaptation to varying workloads [12].

We believe that using an energy management architecture would alleviate or even prevent these types of problems. Section 3 examines in greater depth how applications might benefit from this architecture.

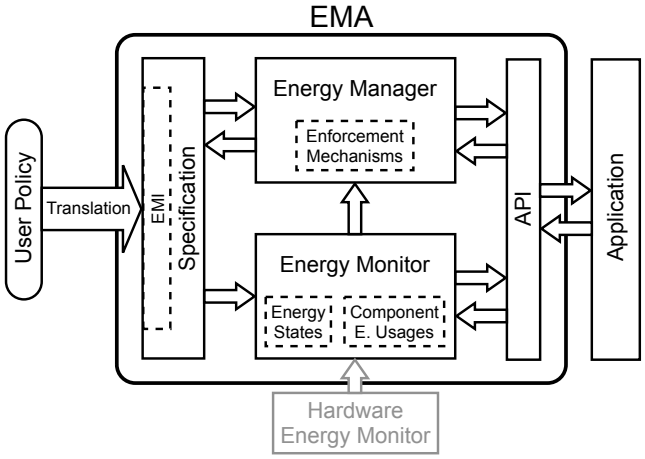


Figure 1. Energy Management Architecture

To provide a basis for energy management, we suggest an architecture that allows sensornet application writers to treat energy as a fundamental design primitive. Building systems that manage energy as a critical resource is not a new concept; research in a number of areas harnesses this idea. In fact, our architecture incorporates many of these concepts, including classifying energy as a first-class OS resource [26], prioritizing resource requests [2], accounting for fine-grained energy consumers [19], allocating resources based on dynamic constraints [7], and providing quality-of-service (QoS) guarantees by using feedback [13]. In addition, we adopt a three component decomposition that is common for architectures managing scarce shared resources, seen in Figure 1: (1) a *policy* interface for user input, (2) a mechanism to *monitor and control* system and component resource usage, and (3) a *management* module for enforcing policy directives. Section 2 describes the specific roles of these components in our architecture.

By employing and adapting concepts from traditional OS, networking, and architecture literature, we envision an energy management architecture (EMA) that allows sensornet applications to accurately view and efficiently manage the energy budget of a node. The primary contributions of this work beyond existing architectural efforts are facilities for: (1) individual accountability for management of com-

putational units relevant to sensornets, (2) priority of policy directives to enable graceful degradation and predictable operation in a dynamic environment, and (3) expression of network-level policy that can be used by individual nodes for local optimization and shared among nodes to adjust the behavior of the network as a whole. In its entirety, the EMA promotes prioritized enforcement of policy during runtime operation as well as enables improved system behavior visualization for debugging during application development.

2 Architecture Overview

Our vision of an energy management architecture consists of three basic components, reminiscent of classic OS resource management architectures: a specification component, an energy monitor, and an energy manager, as seen in Figure 1. The specification component provides a set of interfaces to the programming paradigm and communicates the energy policy to the monitor and the manager. The monitoring component observes granular component energy usages via a set of interfaces to the application. It also monitors overall system energy such as remaining battery energy and incoming harvested energy. The management component uses energy information from the monitoring component to enforce user policies conveyed by the specification component and performs admission control on activities via interfaces to the application similar to the interfaces used by the monitoring component. While some of these ideas have been well-studied in other areas of computer science, they have not been applied to sensornets effectively due to unique resource challenges. We highlight some of these challenges and offer possible solutions.

2.1 Specification of User Policy

Allowing a user to specify a set of directives that are dynamically checked and automatically satisfied at run-time by the system is a powerful idea that exists in many subareas of OS and networking. Applying this principle, we believe that empowering the user of a sensornet with the mechanism to directly specify energy-related requirements is very useful. Following is an example:

```
1. Network lifetime of at least one year.
2. Sample all sensors at 1Hz.
3. a. Communicate readings on multihop tree.
   b. Store readings locally.
4. Maximize sampling rate.
```

2.1.1 Expressive Interface

The specification component in our EMA provides an interface (herein, EMI) to the upper layer programming model. The programming model is responsible for translating user policy, usually written in a high-level language, through use of the EMI. The expressiveness of EMI plays a significant role in both the complexity of EMA and the amount of responsibility required of the programming model.

There are several benefits of having an expressive EMI. A rich interface will allow EMA to support a range of programming models and user policies. If a programming model is relatively simple, it can offload user policy to EMA without performing complicated translation; if a programming model is complex or if it performs some measures of energy management internally, it should still be able to use the EMI effectively while maintaining tight control over resources.

Furthermore, users may desire to specify network-level directives such as total network lifetime or uniform sampling rate. Therefore, EMI should be expressive enough for the user to specify a broad range of network behaviors. Such requirements may be satisfied in a centralized fashion where a node is selected to translate the network-level policy into node-level commands and issue them to the rest of the network, or in a distributed fashion where individual nodes translate the policy and coordinate with other nodes. We favor the distributed approach because, by preserving the high-level policy to the node level, individual nodes have more flexibility and authority to adjust to the locally optimal operating point while still retaining the ability to coordinate with other nodes to reach a global optimum. This is particularly important in a dynamic environment where energy is changing and non-uniform.

However, the expressiveness of EMI needs to be bounded and tractable. For example, we could limit the directives to contain only one optimization clause (maximize / minimize) while the rest of the clauses must be binary predicates (true / false). This, when coupled with strict prioritization, essentially reduces the algorithm complexity to linear time with one variable for optimization.

2.1.2 Priority

User policy is composed of a set of directives, all constrained by a single shared resource – energy. Furthermore, availability of this energy is often unpredictable due to non-uniform communication cost and environmental fluctuations (e.g. energy harvesting). We need some way of coping with the inevitable tension between directives. One solution is to associate directives with priorities, such that the system can degrade in a predictable fashion. This mechanism allows EMA to handle situations when a subset of the requirements cannot be met, which is highly likely.

For example, using the policy shown earlier, the line-item numbers could correspond to the priorities of directives while the sub-numbering (e.g. 3.a, 3.b) indicates to "do one of these," with the higher ones having priority (i.e. *a* before *b*). Using this policy, EMA would first guarantee a lifetime of one year, then a minimum sampling rate of 1 Hz, followed by either sending readings up the tree or storing them locally, depending on whether there is enough energy for sending. If there is more energy, the sampling rate is increased. If the available energy is insufficient to meet all of the directives, the system degrades in the reverse pattern: first lower sampling rate until it reaches 1 Hz; when there is not enough to satisfy 1, 2, and 3.a, do 1, 2, and 3.b.; and so on.

2.1.3 Exception Handler

It is often important for the user to have visibility of network health, especially when the network begins to degrade. For example, it may be important to reserve enough energy to send a packet through the network to the user informing her each time a directive in the policy can no longer be met. We incorporate this idea by introducing an exception-handling mechanism in EMA to include the user in the feedback loop about its status with respect to the policy. If a constraint

can no longer be met, the node can invoke a predefined exception handler. Separate handlers can be defined for each directive, which will be invoked in ascending priority order (low to high). For example, a reasonable exception handler might send a status packet to the base station. The user can then make use of this information by changing the policy or noting it for later use (when analyzing collected data). The exception handler could also be more elaborate, such as requesting node neighbors to take on increased responsibility or issue a new policy. A general exception-handling mechanism gives the user flexibility to control exactly what happens when the network degrades.

2.2 Energy Monitoring

An accurate runtime view of the energy states is necessary for the EMA management component and can be very useful to applications, routing protocols, MAC protocols, or any other components performing energy-related optimizations. Similar to traditional OSes, presenting energy information at the OS-level usually provides reusability, interoperability, and efficiency. We divide energy information into two parts: component energy usage profiles and system energy levels.

2.2.1 Component Energy Usage Profiles

Conceptually different activities, such as temperature sensing, external storage usage, and radio usage, should be *individually accountable*. This provides the system with a fine-grained view of energy usage and allows for arbitration and admission control of activities based on user policy. The separation of activities ideally correlates with the guidelines in the user policy and therefore provides the energy management component with a basic unit for control.

But unlike in a traditional OS, there is often no clear unit of accounting for energy, such as processes or threads, in sensornets. This is further complicated by cases when one component (e.g. the radio) is consuming energy on behalf of another component (e.g. a light sensor), in which case the energy used by the radio should be logically accounted for as light sensing. There have been several research efforts that attempt to provide some notion of grouping. They may be adapted to providing energy accounting in different ways, varying on flexibility, code reusability, and complexity. The suitable choice would depend on the application and programming paradigm. For example, we may be able to group activities by task and manage the tasks through an energy-aware task scheduler. This grouping is naturally supported by task-based systems such as TinyOS [15] or Tenet [8]. However, in the case of TinyOS, even though it uses task IDs, it does not currently support thread context and therefore it cannot easily differentiate multiple invocations of the same task by different activities. The Tenet task library presents functional blocks at a relatively high level and could be grouped by activity fairly naturally. However, some tasks that are used by multiple activities will require extra attention for differentiation. TinyOS 2 (T2) [14] defines a hardware abstraction layer (HAL) that allows grouping by hardware. However, the granularity of HAL is low-level and may not easily correspond to activities in the energy policy. Similarly, the resource component [12] in T2 provides a fixed grouping, albeit at a higher level than HAL. A more flexible approach is to allow arbitrary grouping by the application via a set of APIs. This approach allows the application to decide what should be logically grouped into an activity. However, this will require rewriting existing applications to use the API.

2.2.2 System Energy Levels

System energy levels refer to the actual energy in the system, such as the remaining battery energy as measured by a hardware monitoring facility (such as [9]) or through software estimations. For example, the current energy level can be calculated in software by assuming an initial energy and subtracting component energy usages. However, software methods usually assume a set of static component energy profiles that may not be accurate, do not account for all possible energy consumers, and accumulate error over time. Hardware monitoring facilities, if available, provide much more accurate energy measurements.

2.3 Energy Management

Similar to traditional QoS in networking, the energy manager uses some form of admission control to regulate component accesses to energy resources, providing QoS based on user policy. However, unlike traditional energy management systems, the scope of our manager extends beyond making local decisions, as there may exist network-wide polices that require some sort of coordination between nodes. In sensornets, energy management deals with both local and global resource management.

Locally, we can use system energy levels and component usage profiles from the energy monitor to make decisions based on user policy. For example, an average consumption rate can be calculated to estimate the remaining lifetime. This estimate, coupled with real-time component energy usage, allows the energy manager to perform per-activity admission control. Globally, we can manage resource usage by sharing system energy levels amongst nodes. This is useful if, for example, the policy requires uniform sampling, as a network can only sample as quickly as the node with the lowest sampling frequency. EMA may also borrow management strategies from networking, such as rate control protocols used for the Internet. This allows fairness to be considered among competing energy-consuming activities.

Distributed coordination raises questions about the architecture. It is unclear where the networking component, used to send messages among nodes, will reside. It can be part of the manager, but must closely interact with the network layer of the system to send and receive coordination packets. It may also be collected into a library that is imported into the image at compile-time if a statement in the user policy requires network-wide coordination. Perhaps it is best suited as an application-specific implementation that is facilitated by providing the necessary interface to the energy states. Since distributed coordination is not always necessary, integrating it into the architecture monopolizes valuable resources. We will concentrate on the advantages of either making this component a compile-time library or providing hooks for implementation in the application layer.

We also consider ways to evaluate user policy in a design-time "sanity check." This may be done by collecting information from the network and comparing the active state of the nodes in the network with a known knowledge base or model to see if the provided user policy is feasible. This could be helpful to the end user in designing policy and ensuring, within a reasonable performance bound, that user requirements can be enforced.

3 Application Examples

This section presents two examples of how specific sensornet deployments could benefit from using EMA. We show that each application is representative of a broad class of applications and that EMA is flexible enough to meet varied requirements while providing specific advantages to application developers. Additionally, by providing EMA with multiple degrees of freedom, such as variable sampling rates, reporting rates, and the ability to use low-consumption storage rather than the radio, the application developer can increase the probability of successful deployments.

3.1 Redwoods

In [24], the authors describe a deployment that sought to examine the microclimates surrounding Redwoods trees through measurements of temperature, relative humidity, and incident and light intensity, taken every five minutes. As is common for most monitoring applications, this sampling period was selected through an iterative evaluation between environmental and computer scientists based on the characteristics of the phenomena under observation, the battery size, and the *estimated* energy consumption of each sense, log, and send operation. Readings were logged and reported for a total of 44 days, though some nodes were not functional for the entire duration due to depleted batteries. The cause of this unexpected energy depletion was a software bug that kept nodes in an energy-consuming "listening" state if a communications opportunity (every 5 minutes) was missed. The energy monitoring component could have been used during the application development process to provide per component energy usage statistics. Upon visualizing this unforeseen energy consumption, the designers might have tracked down this bug prior to deployment, averting energy waste and the resulting exhausted nodes and missed data gathering opportunities.

If, for example, energy usage were not tested in advance, EMA could still circumvent the faulty behavior. Below is an example set of user policies for this application.

```
1. Network lifetime of at least 1 month.
2. Sample all sensors every 5 minutes and
   a. Send readings to base station.
   b. Log readings to local storage.
```

The notion of priority in EMA user policy permits *graceful degradation* of these requirements. During the course of the experiment, if the energy monitor calculates that there will be insufficient energy to survive for the rest of the desired lifetime (in this case, one month) while sampling and sending (item 2.a.), the energy manager will abandon item 2.a. and instead enforce items 1 and 2.b. (i.e. sample and store the readings). In the event that there is not enough energy to meet either of these policies, the energy manager will use the remaining energy to perform the exception handler specified by the user. In this way, priority protects a system that is unable to meet all the goals of the user by at least satisfying the most important goals. We argue that using EMA for this application would either allow this fault to be caught in the development process or prevent it from occurring during runtime operation, increasing overall data yield of this application substantially.

These benefits are broadly applicable to a large class of environmental monitoring applications where specifications are strict and static (e.g. sample every 5 minutes) such as [25, 22, 21, 18]. Essentially, EMA facilitates expressing priority in user policy and handling the tradeoffs that inevitably appear in the face of dynamic environmental conditions.

3.2 Arctic Observations

Dr. David Carlson, in his SenSys 2006 keynote speech about the International Polar Year (IPY), urged the sensornet community to develop applications for Arctic observations. He argued that sensornets are a solution to the chief requirement of these applications: broad observation over timescales of an entire season or longer with minimal device maintenance. This requirement arises from the inaccessibility of the environment, often either too remote, expensive, or harsh to visit frequently, and the observation of phenomena with relatively long durations. In these applications, network lifetime is the overriding constraint while, perhaps, sampling and reporting rates may be relaxed.

Consider an application to monitor the melting of glaciers similar to [17]. With a minimum lifetime encoded as the first requirement, reporting and sampling rates can be modulated in order to achieve the foremost requirement of lifetime, as is shown below.

```
1. Network lifetime must be at least 1 year.
2. Sample all sensors at least every 4 hours and
   a. Send readings to basestation once a day.
   b. Log readings to local storage.
```

Network lifetime as the overriding constraint characterizes a class of applications where challenging environments and isolated networks (i.e. no uplink) prevent predictable communication and sensing patterns. In addition to enabling this class of applications, EMA facilitates the class of lifetime-centric deployments in which the phenomena dictates the duration of the network, such as the life of a rat, the pollination cycle of a flower, or the extent of a storm season. Furthermore, by empowering EMA with another degree of freedom, in this case a variable sampling rate, the energy manager component is able to make informed decisions regarding the tradeoffs presented by the user policy and the environmental conditions.

4 Proof-of-Concept Exercise

In this exercise we show some potential benefits of an OS-level energy management service. This is purely a proof-of-concept exercise written in TinyOS. We want to demonstrate that by accounting for individual energy-consuming activities and performing run-time optimizations based on priori-

tized user policy, even a trivial implementation can provide some amount of assurance and predictability in addition to better resource utilization in a dynamic environment.

We use a simple *request/grant* API and provide energy management as a generic TinyOS component (herein EM). We use a uniformly-weighted moving average to estimate current power for a particular activity, hold requests in a 1-deep queue, and grant only when there is enough energy to satisfy both the requested activity as well as all other activities with higher priorities. The application has the flexibility to specify which energy account to debit and by how much using the *request(account, energy_type_string)* command and predefined types. Additionally, the application has the flexibility to group tasks as it desires for the *grant()* event.

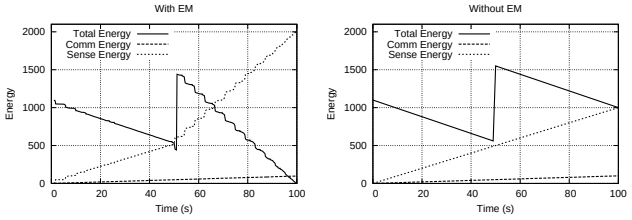
We make the following assumptions: (1) Each mote has an initial energy of 1100 units; (2) Sensing costs 10 energy units per sample; and (3) Communication costs 1 energy unit per packet.

The user policy below is directly encoded in a header file:

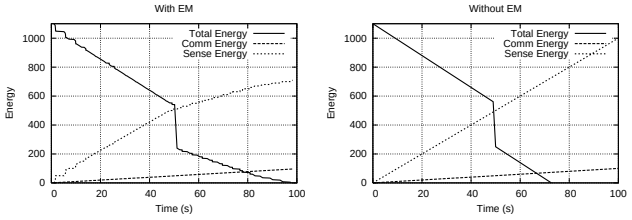
```
1. Network Lifetime >= 100 seconds.
2. Allow all radio traffic.
3. Sense as fast as possible.
```

Three motes are subjected to different (simulated) environmental conditions and their energy states are periodically reported by radio. The communication rate is initially set to 1 packet per second.

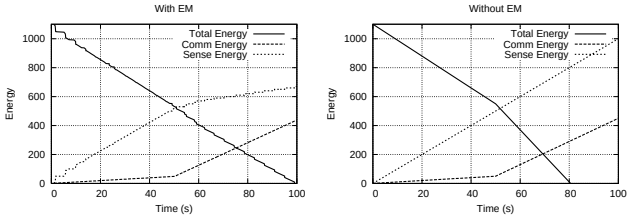
As seen in Figure 2(a), when system energy is increased at t=50 (to simulate energy harvesting), EM automatically increases the rate of grants for sensing, achieving better energy utilization. Conversely, in Figure 2(b), EM reduces the sensing rate when detecting a drop in overall energy at t=50, whereas the mote without EM dies at around t=70. Figure 2(c) shows a plausible scenario where the communication rate increases in the middle of the experiment, possibly due to environmentally-induced retransmissions or increased forwarding responsibility. With EM, the mote is able to reduce sensing and satisfy the lifetime guideline, whereas without EM, the mote dies at t=80. For all EM-enabled motes, the lifetime is exactly 100 seconds with full energy utilization, while the sensing rate is adapted to comply with user-defined policy.



(a) 1000 units of energy are injected at time t=50, i.e. energy is harvested.



(b) 300 units of energy are subtracted at time t=50, i.e. energy storage malfunction.



(c) Communication rate increases to 8 Hz at time t=50, i.e. increased transmissions due to interference or more forwarding responsibility.

Figure 2. Behavior of motes under simulated environments. Left columns shows data collected from three motes written using EM while the right column shows simulated data without EM

5 Architectural Implications

The push for an overall sensornet architecture is not a new one. One particular architecture has been outlined in [4], and a series of architectural components have been proposed in [20, 6, 8]. Additionally, a host of industrial standards/specifications are being developed for either sensornets in particular or low-power wireless devices in general.

One of the basic design principles for EMA is to remain as agnostic as possible to the overall sensornet framework within which it will be situated. As such, rather than create an energy management service that is deeply intertwined throughout the system stack, a cleaner approach is to create a standalone component that only integrates through clearly-defined interfaces and is positioned parallel to the rest of the stack. The important aspect of these interfaces is that they are simple and powerful, yet general enough to be used by a variety of services.

Based on preliminary analysis, we believe our architecture will effectively complement a host of existing architectures and programming paradigms, such as [20, 6, 8, 3, 1]. Although one of our design assumptions is the use of TinyOS, the architecture is easily portable to additional operating systems. We provide a concrete example as a single point in the design space.

Our example system is composed of a modular structure, namely SP [20, 23] and NLA [6], as well as DSN [3], a declarative programming paradigm. EMA sits parallel to the rest of the system. SP serves as the narrow waist, decoupling the network layer from the link layer. NLA is a modular network layer framework that decomposes network protocols into a routing topology, a routing engine, a forwarding engine, and an output queue. NLA sits directly on top of SP and exports a narrow send/receive interface to upper layers. DSN models the sensor node as a query processor, allowing users to declaratively specify programs using rules and predicates. The declarative high-level nature of the user interface of DSN provides a suitable mechanism for the interpretation of user policy, which will consequently by translated into the EMI. DSN also provides the ability to specify these requirements both during compile-time as well as dynamically during runtime, fully utilizing the dynamic capabilities of EMA. Based on these requirements, the energy manager

will determine the admission control policy for the system. EMA may use NLA to communicate nodal energy levels between neighbors and coordinate collective actions for satisfying network level directives.

6 Future Directions

We have presented our vision of an energy management architecture that provides energy as a primitive by implementing services for both energy monitoring and management at the OS-level. Our vision of an energy management architecture is by no means conclusive or definitive. Since there are many differing approaches to managing energy, we simply advocate the need for an energy-centric architecture and present our view of how one might look. However, there remain many open questions; as we continue our work, we intend to solidify design decisions and eventually present a concrete energy management component in the near future. We will evaluate this component by writing representative applications using EMA and comparing them to their current instantiations not using EMA, using metrics such as energy efficiency, robustness, and operation visibility.

An integral part of EMA is the energy monitoring component. While software emulation provides an approximation, hardware will, in most situations, provide much more detailed and accurate energy and power profiles. The SPOT [9] energy metering module provides fine-grained energy information with little software overhead. We plan to release the MicaZ and Telos versions in the near future as well. We also encourage others to design similar monitoring hardware for the myriad platforms available. If our EMA proves to be effective, we hope to disseminate it by bundling it with prevalent distributions such as TinyOS.

For the community, an energy management architecture will aid in future designs of energy-aware application and enable a class of energy-centric applications where lifetime and performance requirements can be specified and controlled under a uniform framework. The monitoring component exposes two previously unavailable streams of information: system energy level and component energy usage. The application can utilize this information in many ways, including energy adaptation, fine-grained component level performance adjustment, intelligent duty cycling, evaluation of low-power designs, and graceful degradation.

The specification component empowers the user and programmer with the ability to directly express energy-related policies such as lifetime and sensing rates. The management component enables the system to enforce this policy. This consolidates many variations of energy management under a unified framework and fuses previous attempts in expressive energy monitoring granularity. An energy management architecture not only benefits applications during runtime, but also helps programmers during the debugging phase. As shown in Section 3.1, the EMA presents increased visibility into the network allowing effective debugging of potential problems. We conclude with a call for community involvement in the development of an energy-centric architecture and applications using such an architecture.

7 References

[1] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. Luo, S. Son, J. Stankovic, R. Stoleru, and A. Wood. Envirotrack: Towards an environmental computing paradigm for distributed sensor networks. *IEEE ICDCS*, 2004.

[2] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. *USENIX OSDI*, 1999.

[3] D. Chu, A. Tavakoli, L. Popa, and J. Hellerstein. Entirely declarative sensor network systems. *ACM VLDB*, 2006.

[4] D. Culler, P. Dutta, C. T. Eee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, , and J. Zhao. Towards a sensor network architecture: Lowering the waistline. *USENIX HotOS*, 2005.

[5] P. Dutta, J. Hui, J. Jeong, S. Kim, C. Sharp, J. Taneja, G. Tolle, K. Whitehouse, and D. Culler. Trio: Enabling sustainable and scalable outdoor wireless sensor network deployments. *IEEE SPOTS*, 2006.

[6] C. T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica. A modular network layer for sensornets. *USENIX OSDI*, 2006.

[7] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. *ACM SOSP*, 1999.

[8] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler. The tenet architecture for tiered sensor networks. *ACM Sensys*, 2006.

[9] X. Jiang, P. Dutta, D. Culler, and I. Stoica. Micro power meter for energy monitoring of wireless sensor networks at scale. *In submission*.

[10] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. *IEEE SPOTS*, 2005.

[11] A. Kansal, D. Potter, and M. B. Srivastava. Performance aware tasking for environmentally powered sensor networks. *ACM Sigmetrics*, 2004.

[12] K. Klues, V. Handziski, D. Culler, D. Gay, P. Levis, C. Lu, and A. Wolisz. Dynamic resource management in a static network operating system. *In submission*.

[13] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE JSAC*, 1996.

[14] 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. *Technical Report TKN-05-007, Telecommunication Networks Group, Technische Universitat Berlin*, 2005.

[15] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for wireless sensor networks. *Ambient Intelligence*, 2005.

[16] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: An acqusitional query processing system for sensor networks. *ACM TODS*, 2005.

[17] K. Martinez, P. Padhy, A. Elsaify, G. Zou, A. Riddoch, J. K. Hart, and H. L. R. Ong. Deploying a sensor network in an extreme environment. *IEEE SUTC*, 2006.

[18] R. Musaloiu-E., A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray. Life under your feet: Wireless sensors in soil ecology. *EmNets*, 2006.

[19] R. Neugebauer and D. McAuley. Energy is just another resource: Energy accounting and energy pricing in the nemesis os. *USENIX HotOS*, 2001.

[20] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A unifying link abstraction for wireless sensor networks. *ACM Sensys*, 2005.

[21] N. Ramanathan, L. Balzano, M. Burt, D. Estrin, T. Harmon, C. Harvey, J. Jay, E. Kohler, S. Rothenberg, and M. Srivastava. Rapid deployment with confidence: Calibration and fault detection in environmental sensor networks. *CENS Tech Report #62*, 2006.

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

[23] A. Tavakoli, J. Taneja, P. Dutta, D. Culler, S. Shenker, and I. Stoica. Evaluation and enhancement of a unifying link abstraction for sensornets. *In submission*.

[24] 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. *ACM Sensys*, 2005.

[25] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. *USENIX OSDI*, 2006.

[26] H. Zeng, C. S. Ellis, and A. R. Lebeck. Experiences in managing energy with ecosystem. *IEEE PerCom*, 2005.