A Modular Sensornet Architecture: Past, Present, and Future Directions

Arsalan Tavakoli[†], Prabal Dutta[†], Jaein Jeong[†], Sukun Kim[†], Jorge Ortiz[†], David Culler[†], Phillip Levis[‡], and Scott Shenker[†]

[†]UC Berkeley EECS Dept.
Berkeley, California 94720
{arsalan, prabal, jaein, binetude, jortiz, culler}@cs.berkeley.edu
shenker@icsi.berkeley.edu

[‡]Stanford CS Dept.
Stanford, California 94305
pal@cs.stanford.edu

1 Introduction

Wireless sensornets provide an unprecedented opportunity to gather huge volumes of data about the physical world around us. Sensornets, however, have severe resource constraints, in terms of power, memory, and bandwidth, which make gathering and processing the data an extremely challenging problem. The first wave of sensornet programmers dealt with these constraints by building tightly-integrated and monolithic system stacks. While the resulting systems were far more energy, memory, and bandwidth efficient than traditional systems, they had two unfortunate properties. First, they were *extremely difficult to program*, as they had to confront myriad new networking and sensing challenges and doing so required very low-level control. Programming entire systems using low-level C code is a daunting task for experienced programmers, and all but impossible for the intended users of sensornets: general scientists. Second, the implementations provided few clearly defined internal interfaces, leading to *limited code reuse and interoperability* between applications from various programmers.

In order to deal with these challenges, Culler et al. proposed an overall sensornet architecture [8]. Such a modular architecture decomposes the system into a set of services, specifies a set of interfaces to these services, and can define its own protocols, including packet formats and communication exchanges. Although the Internet architecture provides inspiration, its applicability is limited in this context by the substantial differences between the Internet and sensornets.

After two years of progress filled with successes and failures, this paper takes a step back, and reflects on the evolution of our architectural outlook into its present day form, drawing from our design experiences. We begin in Section 2 by briefly reviewing the history of sensornet software development, ending with a clear articulation of our overarching goal for a networking software architecture for sensornets. We continue on by examining the modular approach to architecture design, and outline four concrete design goals that will guide our future architectural work. In Section 3 we briefly review code deliverables such as SP [22, 24], a link layer abstraction, and NLA [10], a network-layer modular decomposition. We present them as architectural case studies for understanding the rationale behind design decisions made, and determining the causes of our hits and misses. Section 4 continues by examining the implications of these new design goals on the Berkeley modular architecture. Finally, we conclude in section 5 by exploring future directions.

2 Design Goals and Principles

As the NEST project and its support of TinyOS drew to a close, we observed that rather than being based on large and interchangeable libraries of components, TinyOS systems were for the most part vertical designs. Each major deployment and software tree – Crossbow's software, UCLA's sensor systems, Ohio State's ExScal project, and Berkeley's demos, to name a few – had many small decisions deep within the system based on subtlely incompatible assumptions that made reusability without careful reexamination prohibitively difficult. For example, the TinyOS implementation of diffusion routing [16] was built on top of SMAC [25], which provides a different programming interface than the standard TinyOS data-link layer. Using diffusion therefore required either using SMAC, porting to a non-SMAC interface, or writing an SMAC-AM translation layer. Furthermore, experience showed that tightly coupled code often makes implementation assumptions, which in the general case preclude porting or translation layers.

TinyOS has allowed a large number of institutions to design and implement a wide spectrum of novel and compelling network technologies, but very few of the resulting components work together. Groups with many developers – such as UCLA and Berkeley – developed large suites of interoperable components, but these suites were still incompatible with each other and smaller, more directed efforts (e.g., Fusion [15]) were difficult to incorporate and maintain.

This observation led us towards a design fostering software compatibility and interoperability. To some degree, we knew what we wanted, but needed a several trials (and errors) to clearly articulate it. The core goal of the networking software architecture has been to

> "Enable and foster a large number of diverse networking protocols that can be quickly incorpo-

rated into an application without a large performance cost."

To some degree, many of TinyOS's difficulties in achieving this goal can be traced to imprecise and incompletely specified interfaces. For example, some communication abstractions enforce single outstanding split-phase operations (e.g., GenericComm) while others allow multiple concurrent operations (e.g., QueuedSend). Solving this problem is mostly a question of software design and engineering. However, with the experience gained from five years of research by thousands of users who have developed hundreds of protocols, simultaneously revisiting the abstractions was clearly beneficial. We concluded early on that the diversity of network protocols required a modular approach. Rather than establish a fixed set of protocols, we would provide a toolkit – with a few example uses – with which users could more easily build protocols.

Modular approaches have proven successful in numerous areas. Click [18] decomposed routers into fine-grained modules that could be chained together to implement services. Flux [12] provides a componentized approach to building operating systems for multiple environments. Finally, Exokernel [11] is an operating system architecture that provides application-level resource management by using a library of components utilizing a low-level narrow interface. These three architectures were relatively successful, in terms of being adopted by the community. However, the key observation that is shared among these three frameworks is the maturity of the target field. Routers, operating system kernel design, and resource management were all reasonably well understood problem domains, and hence building a modular architecture to fit the outstanding need was more easily possible. While gradual innovation certainly continued, the system as a whole remained relatively stable.

Sensornets, on the other hand, have not reached this level of maturity. There continues to be large paradigm shifts, and even a specific target application with performance requirements is not clearly specified. Since we began working on the software architecture, new requirements have emerged. Three, in particular, have broad reaching implications we had not considered:

1. Storage-centric networks [21]
2. Packet rate control [23]
3. Highly-tunable bit-rate control [9]

The rapid proliferation rate of new fundamental services has made specification of modular components and strict interfaces difficult and often quickly outdated. The large number of emerging architecture and protocols specifications, both academic and industrial [2, 5, 1, 3, 4], exemplify this uncertainty. However, we argue that sensornets are in the midst of an era of innovation and evolution, and these developments must be fostered and carefully cultivated, rather then confined by imposing premature modular architectures and standards.

Based on these observations, we propose a new concrete set of design principles to guide the development of elements of the networking software architecture. In order, these goals are:

1. Collaborative use
2. Completeness
3. Extensibility
4. Code Reuse

The first goal, collaborative use, addresses the fundamental problem encountered in early TinyOS systems. Two different protocols need to be able to collaboratively share underlying resources. Highly tuned research protocols that are designed to work in isolation can show the limits of what is possible, but for them to be practically useful in pushing technology forward, there must be implementations that play nice with others.

The second goal, completeness, is the one that has been the major cause of difficulty. Being general enough to allow most if not all protocols to be implemented is feasible at any point in time, but as systems evolve and new mechanisms emerge, this can be difficult. Similarly, mechanisms which seemed useful historically may become unnecessary in the future.

The third design goal tackles this tension, embodying our main argument that the focus of the architecture is supporting the innovation and evolution in sensornet research, in order to help drive the field towards maturity. While code reuse and independent development are important considerations, their utility is limited by the constant flux of the overall system. Finally, co-existence of systems will be of increasing importance in the future, but not until standards and unified architectures become accepted by the community.

We note that this list of design goals is not expected to remain stable over time. As the field matures, the importance of encouraging evolution and innovation diminishes, making way for the other, more traditional, goals such as code reuse and interoperability. We envision this maturity occurring from the bottom-up, meaning that the lower parts of the system stack will embrace a modular approach much sooner. TinyOS 1.x, as well as T2, both provide a modularized operating system kernel. Furthermore, the role of SP has gradually become better-defined, and the hope is that the period between version releases will be increasing over time.

In the next section, we describe two major components of the architecture, the SP abstraction and the network layer decomposition, describing how they follow the principles and how the evolution of networking has forced us to reconsider assumptions made in their design.

3 Case Studies

Beginning with the DARPA NEST project, a major research goal at UC Berkeley has been to provide flexible, efficient, and reusable software architectures for sensornet researchers. The TinyOS operating system, for all of its limitations, has proven to be customizable and flexible enough to be beneficial to many research efforts. Furthermore, while difficult at times, being in the position of providing a software system for a diverse set of cutting-edge users requires understanding their needs and therefore, more broadly, the requirements that sensornets introduce.

The growing complexity and interest in sensor networking led us to, over the past few years, design and build a

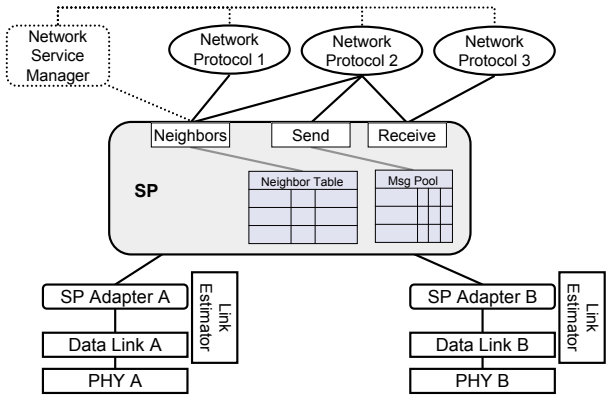

Figure 1. The SP Abstraction

modular software architecture for network protocols. In this section we focus on our two main architectural components, SP [22] and NLA [10]. We examine the design progression of our work toward a modular software architecture for network protocols, using these two components as case studies. We describe the motivation behind certain design decisions, some of which have been very successful, others less so.

3.1 SP

The observation that the single-hop layer in TinyOS (Active Messages) is generally quite stable and used with little modification led us to the conclusion that a single-hop protocol forms the unifying low-level programming abstraction for sensornet protocols [8]. However, while active messages were often used as designed for isolated experiments of a single network protocol, experience with larger deployments showed us that arbitrating between multiple network protocols was a critical requirement. SP was designed to serve as the architectural narrow-waist, effectively decoupling development of network protocols from underlying link layers, and managing co-existence of multiple protocols, while continuing to export a single-hop programming abstraction. Although other components that separated the network and link layers had previously been proposed [14], they focused on specific functionality, such as data aggregation, rather then overall architectural design.

One of the major challenges faced by SP was providing a uniform abstraction for widely different MAC layers. For example, TDMA MAC layers inherently limit the set of reachable nodes but in theory provide interference-free slots, while CSMA layers provide easy connectivity but encounter many more hidden terminals.

Figure 1 provides an overview of the SP abstraction. SP provides two key mechanisms to bridge between network and data link protocols. A shared single-hop neighbor table allows network and data link protocols to share connectivity information, while a message pool can batch transmissions from multiple protocols, thereby taking advantage of TDMA slots or amortizing the cost of an expensive wakeup packet. Finally, link adapters are used to provide a standard interface for all underlying link layers.

Having run and implemented a variety of protocols and applications on top of SP we gained some insight on which features work well, which are superfluous, and which are missing. In some cases our observations conflict, due to differing requirements of upper-layer protocols. For example, message futures are excellent for single-hop protocols, as SP can minimize active time by sending tight packet bursts. However, for flow-based multihop protocols that seek to avoid interference in CSMA networks, such as IFRC [23], SP's packet scheduling prevents specifying maximum packet rates.

Among the successful aspects of SP is the ability to seamlessly run multiple concurrent protocols and services, meeting the first design goal. Rather than have a shared packet queue for network protocols (e.g., TinyOS 1.x's QueuedSend), in SP each protocol has a single outstanding packet and tells SP how many packets it would like to send (message futures). SP's abstraction naturally fits concurrent models, even providing the opportunity to implement specific policy managers that dictate system-wide, rather than protocol-wide, policies. The importance of such abstractions are underlined by the trend towards increasingly complex sensornet systems. SP also satisfies the fourth design goal in that it reduces the code and memory footprint of complex systems.

When SP was designed, it satisfied the second design goal, completeness. While we did not write full implementations for protocols such as SMAC or PSFQ, discussions led us to believe that doing so would not be problematic. However, as new protocols emerged, they introduced new requirements which SP could not satisfy. When developing SP the idea of having SP schedule packet transmissions to maximize energy efficiency seemed a tremendous benefit, but as the importance of rate-limiting, to avoid self-interference, in CSMA networks has become clear this design decision has become a liability. Transport protocols such as IFRC [23] that require controlling timing do not find the required mechanisms in SP. SP support for service-specific architectures, such as energy management and resource arbitration, was missing. On the link side, SP was to some extent biased towards 802.15.4 and the CC2420, ignoring capabilities such as radio bit-rate control [9]. Furthermore, the exported interfaces provided limited access to cross-layer services such as discovery and time synchronization. Support for such features would not only require additional interfaces, but also new internal mechanisms as well. When designing SP we realized that support for such services and programs might eventually be necessary, but we did not realize that their exclusion would cause problems in the near term. We return to this theme in the next section.

Conversely, there were features that seemed appropriate initially but remain largely unused. The predominant example of this involved the feedback mechanisms used by SP to communicate with the upper layers. Although the information was semantically meaningful at lower layers, the feedback was not useful for most network protocols and applications. This experience highlighted the need to work in a top-down fashion, flushing out the requirements of the upper layers, rather than building into SP the set of services that seem useful or practical. In other words, the current state of sensornet research necessitates that SP provide a link

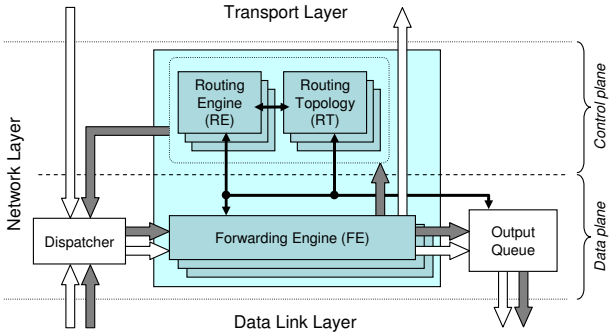

Figure 2. The Network Layer Architecture

abstraction that supports existing implementations, more so than building a narrow waist that exports a set of services that new implementations are expected to adhere to. While future applications may use such features, the disregard for these interfaces by current protocols, coupled with the desire to maintain a lean narrow waist, leads to a top-down focus.

In designing the unifying link layer abstraction for T2 [20], our approach has been quite different. We began with a foundation similar to the previous version, pruning unneeded services and functionality. More importantly we synthesized a list of requirements for a diverse set of protocols and applications in order to ensure support in the upcoming version. We also focused on integrating cross-layer services, such as security and power management, working closely with developers of these stand-alone components and focused architectures. The result is a new link abstraction which we feel is lean, yet provides the essential set of services needed to support the majority of higher-level services.

3.2 NLA

The network layer architecture component [10] was designed to extend the modular architecture upward, building on top of SP. From a very early stage, it was apparent that a single network protocol or standard could not be enforced. Unlike IP's point-to-point best effort service, sensornets utilized a variety of communication paradigms, none of which could single-handedly support the needs of all applications. As a result, the goal became to develop a framework for implementing network protocols, focusing on maximizing code reuse by identifying shared functionality across protocols. Additional functionality was added to the framework, such as a buffer manager and support for managing multiple protocols simultaneously(at a higher level than SP), although certain features such as link estimation and neighbor table management were pushed down to SP. The core of NLA is the decomposition of protocols into four distinct components: a routing topology, a routing engine, a forwarding engine, and an output queue. Figure 2 provides a graphical overview of NLA.

The initial implementation results were encouraging. Five different protocols were successfully implemented, often running concurrently on a single mote. Furthermore, the exercise validated NLA and SP's ability to work together and successfully develop a stack. Performance results were acceptable as well. Five of these six protocols were developed at Berkeley, and while on the one hand it meant that we were experts in analyzing their requirements, it increased the likelihood of failing to support other classes of protocols.

However, a series of challenges soon arose, some implementation specific, others a result of the NLA modularity itself. The battle between two diametric approaches, a rigid framework with generalized interfaces or a flexible architecture with loosely-defined interfaces and horizontally integrated components, embodied the difficulty of designing a modular network layer. In order to maximize code reuse and independent development, the rigid approach clearly specified the functionality present in each component, and provided generalized interfaces between the components to allow for arbitrary pairings. Furthermore, a single, standard interface was exported to higher layers.

Such an approach raises certain issues. First, the increased level of generality increases code size and reduces performance, as these general interfaces are essentially wrappers for protocol specific components. Second, and more importantly, being able to arbitrarily connect components may not be desirable. For example, while a tree routing topology connected to a geographic routing engine may compile, it is semantically meaningless. Furthermore, selecting a single interface exported by the network layer is difficult, as different applications and transport layer protocols have widely varying requirements and interaction mechanisms. Flush [17] serves as an example here, requiring rate-controlled sending and node-wide transmission suppression, neither of which is typically required by other protocols. Finally, the restrictions of such an approach limit the set of protocols that can be implemented. There are certain protocols that do not cleanly decompose into these four components, or that have specific requirements in terms of header format and processing order, like IP, and as a result can not be implemented within this rigid framework. On the other hand, while it is true that a more flexible architecture avoids many of the pitfalls of the first approach, it has one major shortcoming: it limits code reuse and hampers independent development, and moves back in the direction of monolithic implementations.

One might argue that such challenges can be overcome with a better design. However, we feel that NLA is a solid instance of a modular network layer, and much like SP, after a second version is released based on feedback and lessons learned, will be a highly effective tool that will become an integral part of our sensornet architecture. At the same time, we can not ignore the fundamental issues discussed in this section, notably that a modular network layer may not be the optimal solution in all cases. We return to this discussion of evaluating the effectiveness of a modular approach to building a sensornet architecture in Section 4, but first take a closer look at the design goals and principles of a sensornet architecture.

3.3 Overview

In all three architectural steps – original vision to SP, SP to NLA, and NLA to network protocols – we have encountered oversights which have forced us to revisit our decisions and

designs. For example, the original vision for the network architecture called for the unifying protocol to be a single-hop broadcast with arbitrary reception predicates. However, when developing SP we realized that this was not efficiently achievable with TDMA layers such as 802.15.4, forcing us to revisit our approach.

4 Architectural Implications

The discussion of design goals and guiding principles has a profound impact on the Berkeley Modular Sensornet Architecture and its future. The importance of innovation and flexibility, particularly at the higher layers of the stack, coupled with the limited support for these needs provided by a modular approach, render a completely modular sensornet architecture impractical for the moment: too much too soon.

However, modularity maintains its core benefits, and these have proved a good fit for the low-level components of the system. The most important aspect when designing these components is to provide support for the current set of requirements. For SP, two of the main challenges have been integrating cross-layer services, such as real-time control and reliability, cleanly within the system. Furthermore, a broad set of network protocols must be supported, allowing the majority of developers to reliably build on top of this link abstraction. The success of such a component is based on two-part strategy: when the next version of SP is released, it must address a majority of system demands posed by upper-layer protocols, allowing for relatively seamless migrations of these applications to SP-supported system stacks. Consequently, when the system is embraced by the community, it will enable future developers to design their programs around SP, ensuring compatibility. While inevitably new requirements will arise in the future, potentially motivating another release of SP, we see this part of the architecture moving steadily towards stability.

Looking at the upper layers of the system stack, as described in the previous section, the requirements are still somewhat nebulous. New applications are consistently appearing that challenge some of the fundamental assumptions under which we operate in regards to sensornets. Consequently, these layers need the flexibility and extensibility to foster innovation and evolution, and therefore a modular framework no longer seems completely satisfactory. Rather an approach is needed that allows for changing requirements and rapid-prototyping, yet still maintains the needed power of expressivity.

The important question then becomes where in the stack to draw this division. Let us first introduce the taxonomy that will be used throughout this section. We define this lower layer, the modularly constructed aspect of the system, as the *underlying infrastructure*. Examples of this are our SP and NLA components, as well as some high-level components such as Flush [17] in rare cases. We refer to the partial system that sits on top of this underlying infrastructure as a *programming paradigm*. Programming paradigms vary widely, as in essence they are just tools for allowing the end-user to program the network. Examples include DSN (declarative sensor networks) [7], EnviroTrack (middleware for writing tracking applications) [6], Tenet (a multi-tier programming philosophy) [13], and Mate (a virtual machine for sensornets) [19]. Each of these paradigms provides a different programming abstraction for the developer, but it is important to note that all rely on an underlying infrastructure.

Our current viewpoint is that the division between the underlying infrastructure and the programming paradigm should lie around NLA and the network layer, either directly above or below, or perhaps even a hybrid of the two. Despite some shortcomings, we believe that SP has justified its inclusion in any future architectural framework. However, as mentioned in section 2, the difficulties associated with modularity began to surface with the introduction of NLA. NLA provided a framework for implementing a broad set of existing and future network protocols, but the rigid structure also limited expressivity and flexibility. Consequently, we liken NLA's role in the architecture to that of TCP in the Internet Architecture. A large number of applications build on top of TCP as a transport protocol, but for the class of applications where this is not desirable, programming is done directly over raw UDP, which we liken to wiring the programming paradigm directly into SP. Consequently, a mechanism that provides the developer with the flexibility to move the dividing line over time seems to be the ideal hybrid approach.

Returning to the programming paradigms, each is suited for a particular style of programming, often for specific classes of applications, and together they provide varying levels of expressivity, efficiency, and flexibility. As an experiment for testing this hybrid approach to system building, we have been working with the SP/NLA and DSN combination, which we report on elsewhere. This combination appears to work well, but in this section we focus on what specific characteristics of a programming paradigm are important in order for the hybrid approach to succeed.

One of the key goals of this hybrid approach is providing ease of programming for the end-user. As such, programming paradigms which are able to decouple the application logic from the actual implementation are desirable; declarative programming models provide a good mechanism for achieving this decoupling. Innovation and flexibility helps the hybrid approach move the sensornet research forward, but at the same time the power of expressivity must be roughly comparable to the existing approach. Finally, if this high-layer component can provide seamless integration with the underlying structure as the interface, and even the division, shifts then this hybrid architecture can naturally move toward a stable architecture that finds a balance between the upper and lower stack. Furthermore, if the programming paradigm itself can optionally provide modularity then this transition will proceed with even greater smoothness.

So the natural question that arises is what do these observations mean for the Berkeley Modular Sensornet Architecture? Given the current level of innovation, evolution, and paradigm shifting of sensornets, a completely modular architecture is not feasible, or at the very least not completely practical. However, we have instead provided an alternative system, one that utilizes a hybrid approach in which a modular underlying infrastructure is merged with a declarative programming paradigm. The modular lower stack allows for code reuse and independent development, while the upper

layer facilitates innovation and rapid-prototyping.

5 Looking Forward

Synthesizing the design goals and guiding principles is an important preliminary step in the process for creating an architecture. After being faced with difficulties when trying to specify an entirely modular sensornet architecture, we stepped back and analyzed these challenges, looking to draw high-level conclusions regarding the future direction of our work. After making the observation that sensornets are still a research field bustling with innovation and lacking the maturity and stability of other systems, such as query processing and operating systems, we outline a revised set of design goals for our sensornet architecture. With these goals, we come to the realization that a completely modular architecture is too much, too soon. Consequently we propose an alternative hybrid system that better fits the needs of the sensornet community at this current juncture.

We are currently working on the two components of this hybrid approach, the infrastructure services and the programming paradigm. The new version of our unifying link abstraction is under development, being designed for T2. It focuses on fitting the needs of a wide variety of existing applications and protocols. After the link abstraction has been completed, the feedback from the initial version of NLA will be used to design the next-generation modular network layer. The current network layer implementation of T2 already embodies some elements of the NLA modular work.

Our work on the programming paradigm has focused on DSN, which describe in greater detail elsewhere. However, our belief in this hybrid approach is not limited to any one specific programming paradigm, but rather is based on a more general sense that at this time we cannot extend the modular approach much higher than NLA and must rely on other techniques for the high-level programming of sensornets.

Looking to the future, components such as SP and NLA, as well as eventually an entire, integrated architectural framework, will hopefully grow their user base, allowing for a more extensive and updated evaluation of sensornet design principles and requirements. Increased deployments and a gradual maturing of sensornet research will also help further crystallize and stabilize the development of this sensornet architecture.

6 References

[1] 6lowpan: Overview, assumptions, problem statement and goals. `http://www.ietf.org/internet-drafts/draft-ietf-6lowpan-problem-05.txt`.

[2] The epcglobal architecture framework. `http://www.epcglobalinc.org/standards/Final-epcglobal-arch-20050701.pdf`.

[3] Isa-sp100.11 call for proposal: Wireless for industrial process management and control. `http://www.isa.org/filestore/ISASP100_11_CFP_14Jul06_Final.pdf`.

[4] Isa-sp100.14 call for proposal: Wireless network optimized for industrial monitoring. `http://www.isa.org/filestore/ISASP100_14_CFP_14Jul06_Final(2).pdf`.

[5] Zigbee specification, version 1.0. `http://www.zigbee.org/en/spec_download/download_request.asp`.

[6] 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.

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

[8] D. Culler, P. Dutta, C. T. Ee, 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.

[9] H. Dubois-Ferriere, R. Meier, and L. F. P. Metrailler. Tinynode: A comprehensive platform for wireless sensor network applications. *IEEE SPOTS*, 2006.

[10] 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.

[11] D. R. Engler, M. F. Kaashoek, and J. O. Jr. Exokernel: An operating system architecture for application-level resource management. *ACM SOSP*, 1995.

[12] B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, and O. Shivers. The flux oskit: A substrate for kernel and language research. *ACM SOSP*, 1997.

[13] 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.

[14] T. He, B. M. Blum, J. A. Stankovic, and T. Abdelzaher. Aida: Adaptive application-independent data aggregation in wireless sensor networks. *ACM Transactions on Embedded Computing Systems*, 2004.

[15] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. *ACM Sensys*, 2004.

[16] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *ACM TON*, 2002.

[17] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica. Flush: A reliable bulk transport protocol for multihop wireless networks. *In submission*.

[18] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM TOCS*, 2000.

[19] P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks. *ASPLOS*, 2002.

[20] 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.

[21] M. Li, D. Ganesan, and P. Shenoy. Presto: Feedback-driven data management in sensor networks. *USENIX NSDI*, 2006.

[22] 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.

[23] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis. Interference-aware fair rate control in wireless sensor networks. *ACM SIGCOMM Computer Communication Review*, 2006.

[24] 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*.

[25] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. *IEEE INFOCOM*, 2002.