Exposing Resource Tradeoffs in Region-Based Communication Abstractions for Sensor Networks

Matt Welsh
Harvard University
mdw@eecs.harvard.edu

Abstract

We argue that communication abstractions for wireless sensor networks should expose the tradeoff between accuracy and resource usage, allowing applications to adapt to changing network conditions and tune energy and bandwidth requirements. We describe *abstract regions*, a family of spatial operators that capture local communication within regions of the network, which may be defined in terms of radio connectivity, geographic location, or other properties of nodes. Abstract regions provide feedback on the quality of collective operations, and expose an interface for tuning resource consumption. We present the implementation of abstract regions in the TinyOS programming environment, as well as preliminary results demonstrating their use for building adaptive sensor network applications.

1 Introduction

Sensor networks are an emerging computing platform consisting of large numbers of small, low-powered, wireless "motes" each with limited computation, sensing, and communication abilities. Sensor networks are being investigated for applications such as environmental monitoring [4, 17], seismic analysis of structures [3, 12], and tracking moving vehicles [18]. Still, sensor network programming is incredibly difficult, due to the limited capabilities and energy resources of each node as well as the unreliability of the radio channel.

As a result, application designers must make many complex, low-level choices, and build up a great deal of machinery to perform routing, time synchronization, node localization, and data aggregation within the sensor network. Little of this machinery carries over directly from one application to the next, as it encapsulates application-specific tradeoffs in terms of complexity, resource usage, and communication patterns. Many novel algorithms and communication paradigms have been developed for sensor networks, such as directed diffusion [11], TinyDB [15], and GHT [19]. Though each of these mechanisms has a wide range of internal parameters that affect resource usage, these "tuning knobs" are typically not exposed to applications. Our goal is to simplify sensor network development by providing communication abstractions that expose the inherent tradeoffs between resource consumption and accuracy.

In this paper, we describe *abstract regions*, a programming model that supports collective communication over network regions defined in terms of radio connectivity, geographic area, or other properties of nodes. In addition to providing a flexible means of node addressing within local neighborhoods of each node, abstract regions support sharing of data using a tuple-space-like programming model as well as efficient aggregation over shared variables. Abstract regions support adaptivity to changing network conditions by providing feedback on the accuracy of collective operations as well as exposing a set of tuning knobs that control resource usage.

Abstract regions are general enough to support a wide range of sensor network applications and form the basis for other, higher-level communication models. In this paper we motivate their use through two application examples, and describe several implementations, including geographic and radio neighborhoods, spanning trees, and an approximate planar mesh. We present preliminary results highlighting the resource consumption/quality tradeoff in abstract regions and their use for enabling adaptive application design.

2 Motivation and Background

Sensor networks have attracted increasing interest from research and industry. The potential to instrument the physical world at high resolution and low cost opens up a wide range of novel applications in areas such as engineering [3, 12], biology [4, 17], and medicine [5, 21]. The future success of sensor networks depends to a large extent on the programming and communication abstractions presented to application developers.

The bandwidth and energy limitations of sensor nodes typically require that in-network processing be performed to reduce the amount of data that must be transferred out of the network. Application designers are therefore faced with the problem of decomposing an initially straightforward data-collection task into a parallel program with local communication among sensor nodes. Unlike traditional distributed computing environments, sensor networks do not have the benefit of reliable, any-to-any communication channels. Moreover, sensor networks may exist in highly volatile environments, resulting in frequent node and communication failure. As a result, it is often desirable to consume fewer resources (e.g., energy or radio bandwidth) to obtain an approximate result, rather than pay an arbitrary resource cost for complete accuracy.

Communication performance depends on a number of factors, including node density, radio channel quality, and local activity within the network. Moreover, these factors are generally not known *a priori* and may be highly dynamic. Given the diverse needs of sensor network applications, we argue that communication abstractions should expose the quality/resource

consumption tradeoff, allowing applications to make informed decisions as to the cost of operations. This approach makes applications part of the control loop, rather than hiding adaptive logic within underlying layers.

Two important issues are how to define the set of tuning "knobs" and how feedback is provided to applications. For example, an algorithm for computing the maximum value over a set of neighboring nodes may be parameterized in terms of the maximum number of nodes to communicate with, the number of retransmission attempts for each message, and the timeout for waiting for replies from each node. Different settings for these parameters translate into tradeoffs for accuracy, latency, energy consumption, and channel contention. Applications should be given control over these parameter settings and information on their associated resource cost.

A number of systems share our goal of providing a high-level programming framework for sensor networks [7, 8, 11, 19]. Most of these systems provide a fairly broad set of services, and some require that applications conform to a given communication pattern. For example, TinyDB [15] allows data collection queries to be pushed into the network, and is focused on relaying aggregate data along a spanning tree rooted at a base station, rather than performing more general local computations within the network. TinyDB relies on low-level implementations of data aggregation operators to build up queries; one could not easily build an object tracking application directly in TinyDB, unless TinyDB were to provide an object-tracking operator. In contrast, we are interested in providing a lower-level set of communication abstractions that support local processing within the network and can be used to implement higher-level applications and services.

Several adaptive communication mechanisms for sensor networks have been proposed. In many cases, adaptation is hidden within the protocol layer itself, while in others some measure of control is exposed to the application. SPIN [9] is a set of data dissemination protocols that adapt to energy availability by reducing protocol overhead when energy resources are low. TinyDB provides a *lifetime* keyword that scales the query sampling and transmission period to meet a user-supplied network lifetime [16]. Boulis *et al.* [1] describe an aggregation mechanism that exposes an energy/accuracy knob to the user. These approaches are steps in the right direction, and our goal is to identify a unified communication framework that captures these resource tradeoffs at runtime for a broad set of applications.

3 Abstract Regions

Sensor network applications are often expressed in terms of regions over which local sampling, computation, and communication occur. For example, tracking a moving object involves aggregating sensor readings from nodes near the object. Abstract regions are a communication abstraction intended to simplify application development by providing a region-based collective communication interface. Abstract regions capture the inherent locality of communication and hide the details of data dissemination and aggregation within regions.

An abstract region is a collection of sensor nodes with an associated *root node*. Membership within the region is defined in terms of some predicate over each node's relationship to the root. For example, one region definition is "the set of nodes within N radio hops from the root." Other predicates may be used to define regions, such as the set of nodes within distance d. The region defines the notion of the "local neighborhood" around a node, which is useful when performing local, spatial computations within the sensor network. In general, each node in the sensor network defines a set of abstract regions that it wishes to operate over. Note that a node can belong to more than one region at a time: for example, node will belong to several different N-radio-hop regions rooted at different nodes.

Abstract regions support the following set of operators:

Region formation: Before performing other operations on a region, the root first initiates formation of the region, which identifies the nodes that are members. Depending on the type of region, this may require broadcasting messages, collecting information on node locations, or estimating link quality between participating nodes.

Enumeration: The enumeration operator retrieves the set of nodes participating in the region, allowing them to be addressed directly. Supplemental information, such as the location of each node, may be returned as well.

Data sharing: The data sharing operator allows variables, represented as key/value pairs, to be shared amongst nodes in the region. *get(v,n)* retrieves the value of variable v from node n, and *put(v,l)* stores the value l of variable v at the local node. Depending on the implementation, data sharing may involve broadcasting shared variables to the region, gossiping, or separate fetch messages for each *get* operation.

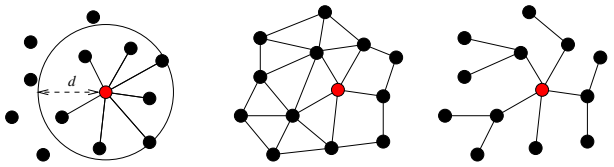
Reduction: The reduction operator takes a shared variable key and an associative operator (such as *sum*, *max*, or *min*) and reduces the shared variable across nodes in the region, storing the result in a shared variable at the root. Alternatively, the result may be broadcast to all nodes in the region.

3.1 Quality feedback and tuning interface

Abstract region operations are unreliable in the sense that they do not guarantee that all potential nodes in the region will be contacted. Region formation and reduction return a *quality measure* that represents the completeness or accuracy of a given operation. The quality of region formation represents the fraction of candidate nodes that responded to the formation request. Reduction quality represents the fraction of nodes in the region that participated in the reduction.

Applications can use this quality feedback to affect resource consumption of collective operations through a *tuning interface*. For example, region formation may be tuned by adjusting the number of messages, amount of time, or number of candidate nodes to consider. Likewise, data sharing and reduction perform message retransmission and acknowledgment to increase the reliability of communication; the depth of the transmit queue and number of retransmission attempts can be tuned by the application.

In our current implementation, the set of parameters exposed by the tuning interface are somewhat low-level and may be specific to the particular region implementation. We are currently working on a resource-centric tuning mechanism that allows the



(a) Geographic region **(b) Planar mesh** **(c) Spanning tree**

Figure 1: *Examples of abstract regions.*

programmer to express an energy, radio bandwidth, or latency budget for each operator, and which maps these constraints onto the appropriate low-level parameter settings. This is discussed further in Section 5.

3.2 Implementations

Given the diverse needs of sensor network applications, we expect a range of abstract region definitions will be useful to programmers. We have completed several abstract region implementations, with several others underway. Three examples are shown in Figure 1. They include:

- *N-radio hop:* All nodes within N radio hops of the root;
- *N-radio hop with geographic filter:* All nodes within N radio hops and distance d from the root;
- *k-nearest neighbor:* k nearest nodes within N radio hops;
- *k-best neighbor:* k nodes within N radio hops with the highest link quality to the root, as measured in fraction of packets dropped over some measurement interval;
- *Approximate planar mesh:* A mesh with a small number (possibly zero) crossing edges; and
- *Spanning tree:* A spanning tree rooted at a single node, used for aggregating values over the entire network.

Abstract regions are implemented in *nesC* [6], a component-oriented programming language for sensor networks using the TinyOS [10] operating system. TinyOS is designed for low-power, resource-constrained sensor nodes, such as the UC Berkeley MICA mote. This device consists of a 4 MHz Atmega128 processor, 128KB of code memory, 4KB of data memory, and a Chipcon CC1000 radio capable of 38.4 Kbps and an outdoor transmission range of approximately 300m. The limited memory and computational resources of this platform make an interesting design point, as traditional programming models (such as threads or complex protocol stacks) cannot be implemented. TinyOS uses an event-driven design and provides an unreliable radio communication stack.

3.2.1 Radio and geographic neighborhoods

The implementation of the radio and geographic neighborhoods is straightforward. Forming the region involves broadcasting node advertisements that the root uses to select a set of member nodes. Data sharing may be implemented using either a "push" (*put* broadcasts updates to neighboring nodes) or "pull" (*get* sends a fetch message to the corresponding node) approach; our current implementation uses the latter. Reduction involves collecting shared variable values at the root and combining them with the reduction operator, storing the result in a shared variable at the root.

3.2.2 Approximate planar mesh

Planar meshes, such as the Delaunay triangulation [20], are useful for spatial computation (e.g., dividing space into nonoverlapping cells), but generally require extensive interprocessor communication to compute. We have implemented an *approximate* planar mesh in which a small number of edges may cross, using only communication within local neighborhoods of each node.

Our algorithm is based on a pruned Yao graph [14] and operates as follows. First, each node forms a k-nearest radio region of candidate nodes. Each node then forms the Yao graph by dividing space around it into m equal-sized sectors of angle $\theta = 2\pi/m$ and selecting the nearest node within each sector as a potential neighbor. Next, each node advertises its set of selected outedges with a single-hop broadcast. Upon reception of an edge advertisement, nodes test whether the given edge crosses one of its own outedges, and if so send an invalidation message to the source, causing it to prune the offending edge from its set. Nodes do not select additional neighbors beyond the initial candidates, so a node may end up with fewer than m outedges. Nodes perform several rounds of edge set broadcasts and wait for some time before settling on a final set of neighbors. Data sharing and reduction are implemented using the same components as in the radio neighborhood, as all neighbors are one radio hop away.

3.2.3 Spanning tree

Spanning trees are useful for aggregating values within the sensor network at a single point, as demonstrated by systems such as TinyDB [15]. The spanning tree region is formed by broadcasting messages indicating the source node ID and number of hops from the root; nodes rebroadcast received advertisements with their own ID as the source and the hopcount incremented by 1. Nodes select a parent in the tree based on the lowest hopcount advertisement they receive. Data sharing is implemented by sending *put* data to the root, which caches the value for future *get* operations. Reduction is performed by flooding a request to all nodes in the spanning tree, causing each node to aggregate its local value with that of its children and propagate the result to its parent.

3.3 Applications

Abstract regions are general enough to support a wide range of sensor network applications, shielding developers from much of the complexity of low-level communication while still exposing meaningful efficiency/quality tradeoffs. In this section we describe two simple applications based on abstract regions: tracking an object in the sensor field and finding spatial contours in sensor readings.

3.3.1 Object tracking

Object tracking is an oft-cited application for sensor networks [2, 22] and involves determining the location and velocity of a moving object by detecting changes in magnetic field. Our version of object tracking uses a simple algorithm[1] in which nodes take periodic magnetometer readings and compare them to a threshold value. Nodes above the threshold communicate

[1]This algorithm is based on one used in the UC Berkeley NEST project demonstration software, which has not yet been published.

with their neighbors and identify the centroid of nearby sensor readings, transmitting the result to a base station.

The following pseudocode shows this application expressed in terms of abstract regions:[2]

```
location = get_location();
/* Get 8 nearest neighbors */
region = k_nearest_region.create(8);

while (true) {
  reading = get_sensor_reading();
  region.putvar(reading_key, reading);
  region.putvar(reg_x_key, reading * location.x);
  region.putvar(reg_y_key, reading * location.y);

  if (reading > threshold) {
    /* Compute ID of the node with the maximum value */
    max_id = region.reduce(OP_MAXID, reading_key);
    if (max_id == my_id) {
      sum = region.reduce(OP_SUM, reading_key);
      sum_x = region.reduce(OP_SUM, reg_x_key);
      sum_y = region.reduce(OP_SUM, reg_y_key);
      centroid.x = sum_x / sum;
      centroid.y = sum_y / sum;
      send_to_basestation(centroid);
    }
  }
}
```

The program performs essentially all communication through the abstract regions interface, in this case the k-nearest-neighborhood. Nodes store their local sensor reading and the reading scaled by the x and y dimensions of their location as shared variables. Nodes above the threshold perform a reduction to determine the node with the maximum sensor reading, which is responsible for calculating the centroid of its neighbors' readings. A series of sum-reductions is performed over the shared variables which is used to compute the centroid, given as

$$c_x = \sum_i R_i x_i / \sum_i R_i$$
$$c_y = \sum_i R_i y_i / \sum_i R_i$$

where c_x and c_y are the (x, y)-coordinates of the centroid, R_i is the reading at node i, and x_i and y_i are the (x, y)-coordinates of node i.

3.3.2 Contour finding

The contour finding problem is expressed as determining a set of points in space that lie along, or close to, an isoline in the gradient of sensor readings. For example, a contour may represent the frontier of an event of interest, such as a thermocline. Contour finding is a valuable spatial operation as it compresses the per-node sensor data into a low-dimensional surface.

Our contour finding application is depicted in Figure 2. Nodes first form an approximate planar mesh region, as described earlier. Each node stores its local sensor reading as a shared variable. Nodes that are above the sensor threshold of interest fetch readings from their neighbors. For each neighbor that is below the threshold, the node advertises a contour point as the midpoint between itself and its neighbor. This results in a small number of contour points advertised along the threshold boundary.

[2]For brevity, the use of tuning and quality feedback is not shown in this example.

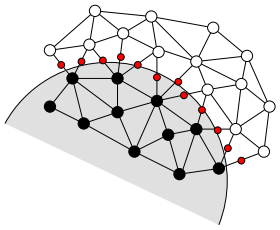


Figure 2: **Contour finding application.** *The shaded region represents an area where sensor readings fall above a threshold. Nodes are connected into an approximate planar mesh, shown as edges between nodes. Each contour point is chosen as the midpoint between a node above the threshold and a node below the threshold.*

4 Evaluation

In this section, we demonstrate use of abstract regions in three scenarios: shared variable reduction, construction of the approximate planar mesh, and tracking a moving object through the sensor field. For reduction, the fundamental resource/quality tradeoff is the number of messages transmitted versus the number of candidate nodes participating in the reduction. For mesh formation and object tracking, performance depends on a number of factors including number of messages sent, timing, and physical density of the sensor network.

These results were obtained using TOSSIM [13], a simulation environment that executes TinyOS code directly; hence, our abstract region code can either run directly on real sensor motes or in the TOSSIM environment. TOSSIM incorporates a realistic radio connectivity model based on data obtained from a trace of radio performance on the Berkeley MICA motes in an outdoor setting. We simulate a network of 100 nodes distributed semi-irregularly in a 20×20 foot area. Because TOSSIM does not currently simulate the energy consumption of nodes, we report the number of radio messages sent as a rough measure of energy consumption. On the MICA platform, radio communication dominates CPU energy usage by several orders of magnitude.

4.1 Adaptive reduction algorithm

Here, we evaluate the use of abstract regions to implement an adaptive reduction algorithm. By performing reduction over a subset of neighbors, nodes can trade off energy consumption for accuracy. As described earlier, the reduction operator provides quality feedback in the form of the fraction of nodes that responded to the reduce operation. Over an unreliable radio link, this yield is directly related to the number of retransmission attempts made by the underlying transport layer, as shown in Figure 3. Moreover, the appropriate retransmission count to meet a given yield is a function of the local network density and channel characteristics, which vary across space and time.

We implemented a simple adaptive reduction algorithm that attempts to maintain a yield target of 75%. The algorithm controls the number of retransmission attempts made by the transport layer with a simple additive-increase/additive-decrease con-

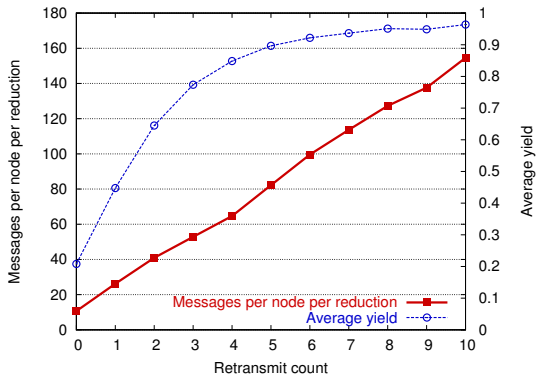


Figure 3: **Reduction yield and overhead in the k-nearest neighborhood region.** *This figure shows the average yield (fraction of neighbors responding to a reduction request) and number of messages for reduction operations. The yield and overhead are directly related to the number of retransmission attempts made by the transport layer.*

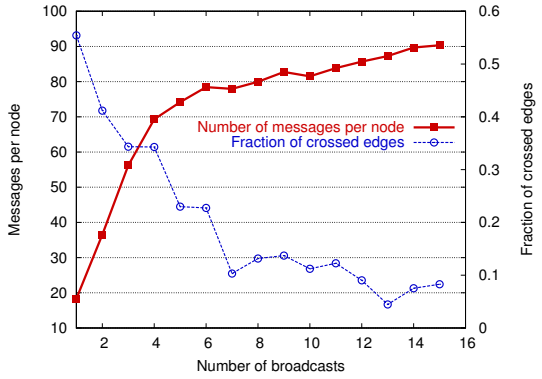


Figure 4: *Quality and overhead of the pruned Yao graph as a function of number of message broadcasts.*

troller. Nodes form a k-nearest-neighbor region (with $k = 8$) and repeatedly perform a max-reduce over a local sensor reading. When the yield is above 85%, the algorithm reduces the retransmission count by 1, and when the yield is below 65%, the algorithm increases the count by 1. Using this algorithm, nodes achieve an average yield (over a sample run of 200 reductions) of 0.778 with an average of 28.3 messages per node per reduction. [3] Each node is tuned to the appropriate number of retransmission attempts that it requires to meet the quality target.

4.2 Approximate planar mesh construction

Constructing an approximate planar mesh is a tradeoff between the number of messages sent and the quality of the resulting mesh, which we measure in terms of the fraction of crossing edges. Given the unreliable nature of the communication channel, our pruned Yao graph algorithm cannot guarantee that the mesh will be planar. For many applications, a perfect mesh is not necessary, since planarity is impossible to guarantee if there is measurement error in node localization.

[3] In the best case, with no lost messages, our reduction algorithm requires 16 messages per node per reduction: one message per neighbor to fetch its value, and one message per neighbor to respond to its reduction request.

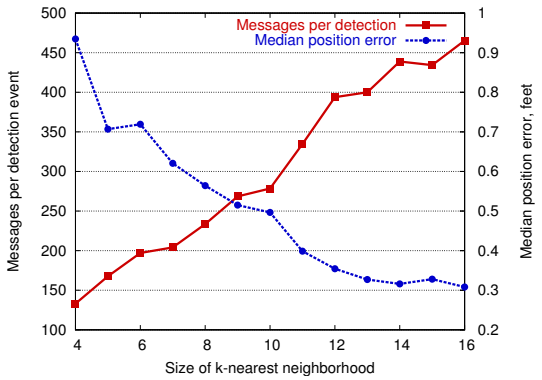


Figure 5: **Accuracy and overhead of object tracking as a function of neighborhood size.**

In our implementation, the quality of the mesh depends primarily on the number of broadcasts made by each node to advertise its location, used for selection by the k-nearest neighbor region on which the Yao graph depends. Figure 4 shows the cost in terms of total messages sent, as well as the fraction of crossed edges, as the number of broadcasts is increased. There is a clear relationship between increased communication and the quality of the mesh. As in the case of reduction, applications can tune the number of broadcasts to meet a given target mesh quality.

4.3 Object tracking accuracy

Finally, we evaluate the accuracy of the object tracking application described in Section 3.3. The application tracks a simulated object moving in a circular path of radius 6 feet at a rate of 0.6 feet every 2 sec. Moving the object in a circular path induces nodes in different regions of the network to detect and track the object. Nodes take sensor readings once a second, the values of which scale linearly with the node's distance to the object, with a maximum detection range of 5 feet.

The resource tuning parameter in question is the size of the k-nearest neighbor region. A smaller number of neighbors reduces communication requirements but yields a less accurate estimate of the object location. Figure 5 shows the accuracy of the tracking application as we vary the number of neighbors in each region. For each time step, we calculate the average distance between the simulated object and the value reported by the sensor network. As the figure shows, as the size of the neighborhood increases, so does the accuracy, as well as the total number of messages sent. Note that increasing the neighborhood size beyond a certain point does not significantly increase tracking accuracy, as more distant nodes are less likely to have detected the object and may not respond to the reduction request due to packet loss.

5 Future Directions

The abstract region is a fairly general primitive that captures a wide range of communication patterns within sensor networks. The notion of communicating within, and computing across, a local region (for a range of definitions of "local") is a useful concept for sensor applications. Similar concepts are evident in other communication models for sensor networks, although

often exposed at a much higher level of abstraction. For example, directed diffusion [11] and TinyDB [15] embody similar concepts but lump them together with additional semantics. Abstract regions are fairly low-level and are intended to serve as building blocks for these higher-level systems.

In the future, we intend to explore how far the abstract region concept addresses the needs of sensor net applications. We are currently completing a suite of abstract region implementations and are developing several applications based on them. We also intend to provide a set of tools that allow application designers to understand the resource consumption and quality tradeoffs provided by abstract regions. These tools will provide developers with a view of energy consumption, communication overheads, and accuracy for a given application. Our goal is to allow designers to express tolerances (say, in terms of a resource budget or quality threshold) that map onto the tuning knobs offered by abstract regions. This process cannot be performed entirely offline, as resource requirements depend on activity within the network (such as the number of nodes detecting an event). Runtime feedback between the application and the underlying abstract region primitives will continue to be necessary.

Finally, we intend to use abstract regions as a building block for a high-level programming language for sensor networks. The essential idea is to capture communication patterns, locality, and resource tradeoffs in a high-level language that compiles down to the detailed behavior of individual nodes. Shielding programmers from the details of message routing, in-network aggregation, and achieving a given fidelity under a fixed resource budget should greatly simplify application development for this new domain.

References

[1] A. Boulis, S. Ganeriwal, and M. B. Srivastava. Aggregation in sensor networks: An energy - accuracy tradeoff. In *Proc. IEEE workshop on Sensor Network Protocols and Applications*, 2003.

[2] R. Brooks, P. Ramanathan, and A. Sayeed. Distributed target classification and tracking in sensor networks. *Proceedings of the IEEE*, November 2003.

[3] Center for Information Technology Research in the Interest of Society. Smart buildings admit their faults. `http://www.citris.berkeley.edu/applications/disaster_response/smartbuil%dings.html`, 2002.

[4] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology. In *Proc. the Workshop on Data Communications in Latin America and the Caribbean*, Apr. 2001.

[5] R. X. Cringely. Chase Cringely: Finding Meaning in a Lost Life. `http://www.pbs.org/cringely/pulpit/pulpit20020425.html`.

[6] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proc. Programming Language Design and Implementation (PLDI)*, June 2003.

[7] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. DIFS: A distributed index for features in sensor networks. In *Proc. the First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[8] J. S. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proc. the 18th SOSP*, Banff, Canada, October 2001.

[9] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proc. the 5th ACM/IEEE Mobicom Conference*, August 1999.

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Proc. the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Boston, MA, USA, Nov. 2000.

[11] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. International Conference on Mobile Computing and Networking*, Aug. 2000.

[12] V. A. Kottapalli, A. S. Kiremidjian, J. P. Lynch, E. Carryer, T. W. Kenny, K. H. Law, and Y. Lei. Two-tiered wireless sensor network architecture for structural health monitoring. In *Proc. the SPIE 10th Annual International Symposium on Smart Structures and Materials*, San Diego, CA, March 2000.

[13] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proc. the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003.

[14] X.-Y. Li, P.-J. Wan, Y. Wang, and O. Frieder. Sparse power efficient topology for wireless networks. In *Proc. 35th Annual Hawaii International Conference on System Sciences*, January 2002.

[15] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proc. the 5th OSDI*, December 2002.

[16] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. the ACM SIGMOD 2003 Conference*, June 2003.

[17] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, USA, Sept. 2002.

[18] K. S. Pister. Tracking vehicles with a uav-delivered sensor network. `http://robotics.eecs.berkeley.edu/~pister/29Palms0103/`, March 2001.

[19] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensornets with GHT, a geographic hash table. Technical Report IRB-TR-03-006, Intel Research Berkeley, March 2003.

[20] J. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):21–74, May 2002.

[21] M. Welsh, D. Myung, M. Gaynor, and S. Moulton. Resuscitation monitoring with a wireless sensor network. In *Supplement to Circulation: Journal of the American Heart Association*, October 2003. Abstract, American Heart Association Resuscitation Science Symposium.

[22] Y. Xu and W.-C. Lee. On localized prediction for power efficient object tracking in sensor networks. In *Proc. 1st International Workshop on Mobile Distributed Computing*, May 2003.