17 Communication Architecture and Programming Abstractions for Real-Time Embedded Sensor Networks[*]

T. Abdelzaher
University of Virginia

J. Stankovic
University of Virginia

S. Son
University of Virginia

B. Blum
University of Virginia

T. He
University of Virginia

A. Wood
University of Virginia

Chenyang Lu
University of Washington at St. Louis

17.1 Introduction

Made possible by advances in communication technology and hardware miniaturization [11], ad hoc wireless sensor networks raise the need for a new suite of communication protocols and new programming abstractions for distributed deeply embedded computing. Such sensor networks are especially useful when an inhospitable, poorly accessible, or delicate environment prevents the installation of needed computing infrastructure; an example would be the site of a natural disaster or a target behind enemy lines. Instead, myriads of tiny, computationally equipped wireless sensor devices may be dropped to form an ad hoc network that operates autonomously to monitor its surroundings, react to distributed events, or alert appropriate authorities when specific activities are observed.

Sensor networks offer new challenges from the perspective of building communication protocols and from the perspective of developing appropriate programming models. These challenges arise due to their large scale, autonomous operation, massively parallel interactions with a spatially distributed physical environment, and a more stringent set of resource constraints.

[*]The work reported in this paper was supported in part by the National Science Foundation under grants CCR-02-05327 and CCR-00-92945; by DARPA under grant F33615-01-C-1905; and by MURI under grant N00014-01-1-0576.

0-8493-1968-4/05/$0.00+$1.50

Communication protocols for sensor networks must provide real-time assurances. Although ensuring proper timing behavior of systems has been a topic of real-time research for decades, sensor network applications offer physical *space*, in addition to time, as a new dimension for interaction with the environment. Thus, while traditional real-time computing research has been concerned with meeting time constraints, a new branch of theory is needed to analyze systems that interact with their surroundings in real time and in the real dimensions of physical space. For example, in a network that tracks vehicles through the sensor field, the application must collect sensory measurements in real time from the actual changing locale in which the vehicle is detected. Message communication must therefore be sensitive to time and distance constraints, which may depend on external factors such as the physical speed of the monitored vehicle. This chapter describes a protocol suite in which time and distance constraints are addressed.

A new programming paradigm is needed to facilitate the task of sensor network application development. Due to the large scale of sensor networks, programmers should not need to be concerned with low-level abstractions and functions such as creating and destroying individual connections between pairs of nodes. Instead, the programming environment must offer a conceptual view in which global tasks can be defined in an abstract manner, leaving it for the underlying system to translate them into computational and communication activities on individual sensor nodes.

This chapter reports on the design of a programming system developed on top of a communication protocol suite that provides the required high-level abstractions. The language allows external events in the environment to be represented as objects in the computing system facilitating the monitoring of such events by the application. The reported architecture is a part of an ongoing research effort to develop a sensor network virtual machine for future distributed deeply embedded applications.

Section 17.2 describes a protocol suite that takes into account time and space constraints and exports a useful transport-layer abstraction in which logical communication end-points can be associated with tracked objects in the external environment. Section 17.3 describes a new programming model for sensor networks that builds upon the aforementioned transport protocol to elevate environmental objects into first-class programming abstractions. Related work is summarized in Section 17.4. The chapter concludes with Section 17.5, which discusses some of the remaining challenges and directions for future research.

17.2 A Protocol Suite for Sensor Networks

Communication protocols in sensor networks are the fundamental cornerstone that glues distributed applications together. The deeply embedded nature of sensor networks presents some of the most interesting challenges in the design of their communication protocols. New research topics span all protocol stack layers, primarily motivated by a tighter interaction between the network and its physical environment. At the MAC layer, new protocols are needed that enforce message priorities consistently with time and distance constraints that arise from environmental interactions [22]. Awareness of the physical environment must also be incorporated into the network layer; for example, location should be an essential attribute of addressable networked objects [15]. Location-assisted routing protocols such as LAR [19] and DREAM [4], as well as location services [21], have been described for ad hoc wireless networks.

More generally, routing algorithms are needed in which destinations are described implicitly by their environmental attributes. For example, directed diffusion [14, 18] and the intentional naming system [3] provide addressing and routing based on data interests. A fundamental rethinking of basic protocols is required at the transport layer as well. Individual socket-style connections between nodes are too low level to be a useful abstraction for the programmer. They must be replaced with higher level alternatives more suitable for the main purpose of sensor networks, namely, monitoring the external surroundings in which they are embedded.

This section describes an answer to the challenge of incorporating environmental awareness into the design of sensor network communication protocols. This protocol stack features two important contributions. First, it implements new real-time message scheduling algorithms in which time and physical distance requirements are observed. Second, it exports a transport-layer address space that associates unique network addresses with external environmental objects. The new addresses serve as connection end-points, thereby raising the level of connection abstraction to entities of direct interest to the application. The layers of this protocol stack are described in the following subsections.

17.2.1 Real-Time Distance-Aware Scheduling

Message communication in sensor networks must occur in bounded time — for example, to prevent delivery of stale data on the status of detected events or intruders. In general, a sensor network may simultaneously carry multiple messages of different urgency communicated among destinations that are different distances apart. The network has the responsibility of ordering these messages on the communication medium in a way that respects time and distance constraints.

A protocol that achieves this goal in this architecture is called RAP [22]. It supports a notion of packet velocity and implements velocity monotonic scheduling (VMS) as the default packet scheduling policy on the wireless medium. Observe that for a desired end-to-end latency bound to be met, an in-transit packet must approach its destination at an average velocity given by the ratio of the total distance to be traversed to the requested end-to-end latency bound. RAP prioritizes messages by their required velocity so that higher velocities imply higher priorities.

Two flavors of this algorithm are implemented. The first, called *static velocity-monotonic scheduling*, computes packet priority at the source and keeps it fixed thereafter regardless of the packet's actual progression rate toward the destination. The second, called *dynamic velocity-monotonic scheduling*, adjusts packet priority *en route* based on the remaining time and the remaining distance to destination. Thus, a packet's priority will increase if it suffers higher delays on its path and decrease if it is ahead of schedule.

To achieve consistent prioritization in the wireless network, it is necessary to have priority queues at nodes, as well as a MAC layer that resolves contention on the wireless medium in a manner consistent with message priorities. A scheme similar to that of Aad and Castelluccia [1] is adopted to prioritize access to the wireless medium. The scheme is based on modifying two 802.11 parameters — the DIFS counter and the back-off window — so that they are aware of priorities. The DIFS counter determines the maximum time a node waits, after the communication channel becomes idle, prior to transmitting an RTS packet. The actual waiting time is randomly chosen between 0 and DIFS. An approximate prioritization effect is achieved by letting the DIFS value depend on the priority of the outgoing packet at the head of the transmission queue.

Because a larger value is given to packets of lower priority, more urgent packets tend to contend on the medium more aggressively. The back-off window of 802.11 increases the maximum waiting time when collisions occur. To give preferential treatment to higher priority packets, this increase is made dependent on the priority of the head of the queue. A higher increase is incurred for packets of lower priority, so collisions tend to be resolved in favor of higher priority packets.

A detailed performance evaluation of this scheme can be found in Lu et al. [22]. It is shown that VMS substantially increases the fraction of packets that meet their deadlines, taking into consideration distance constraints. More accurate schemes for medium access prioritization remain an open research topic. An interesting related topic is that of analysis of VMS. Ideally, such an analysis should allow a source node to determine whether a particular desired velocity is attainable between a source–destination pair, given current network conditions. Although an analytic expression for velocity feasibility is still an open problem, the following subsection describes a feedback-based technique that enforces velocity constraints dynamically by applying back-pressure to slow the sources when such constraints are violated.

17.2.2 Enforcement of Velocity Constraints

Consider a network that supports multiple predefined velocities. An application can choose a velocity level for each message. The network guarantees that the chosen message velocity is observed with a very high probability as long as the message is accepted from the application. A network-layer protocol with the preceding property, called SPEED [13], has recently been developed by the authors. The protocol defines the velocity of an in-transit message as the rate of decrease of its straight-line distance to its final destination. Therefore, for example, if the message is forwarded away from the destination, its velocity at that hop is negative.

The main idea of SPEED is as follows. Each node i in the sensor network maintains a neighborhood table that enumerates the set of its one-hop neighbors. For each neighbor, j, and each priority level, P, the node keeps a history of the average recently recorded local packet delay, $D_{ij}(P)$. Delay $D_{ij}(P)$ is defined as the average time that a packet of priority P spends on the local hop i before it is successfully forwarded to the next-hop neighbor j.

Given a packet with some velocity constraint, V, node i determines the subset of all its neighbors that are closer to the packet's destination. If L_{ij} is the distance by which neighbor j is closer to the destination than i, the velocity constraint of the packet is satisfied at node i if some priority level P and neighbor j exist so that $L_{ij}/D_{ij}(P) \geq V$. The packet is forwarded to one such neighbor nondeterministically. If the condition is satisfied at multiple priority levels, the lowest priority level is chosen. If no neighbor satisfies the velocity constraint, a local deadline miss occurs.

A table at node i keeps track of the number of local deadline misses observed for each velocity level V; this table is exchanged between neighboring nodes. Nodes use this information in their forwarding decisions to favor more appropriate downstream hops among all options that satisfy the velocity constraint of a given packet. No messages are forwarded in the direction of nodes with a high miss ratio. The mechanism exerts back-pressure on nodes upstream from congested areas. Congestion increases the local miss ratio in its vicinity, preventing messages from being forwarded in that direction. Messages that cannot be forwarded are dropped, thus increasing the local miss ratio upstream.

The effect percolates towards the source until a node is found with an alternative (noncongested) path toward the destination, or the source is reached and informed to slow down. The mentioned scheme is therefore effective in exerting congestion control and performing packet rerouting that guarantee the satisfaction of all velocity constraints in the network at steady state [13]. The protocol is of great value to real-time applications in which different latency bounds must be associated with messages of different priority.

17.2.3 Entity-Aware Transport

Although RAP and SPEED allow velocity constraints to be met, the abstractions provided by them are too low level for application programmers. A transport layer is developed whose main responsibility is to elevate the degree of abstraction to a level suitable for the application. In particular, a transport layer is proposed in which connection end-points are directly associated with events in the physical environment.

Events represent continuous external activities, such as the passage of a vehicle or the progress of a fire, in which an application might be interested. By virtue of this layer, the programmer can describe events of interest and logically assign "virtual hosts" to them. Such hosts export communication ports and execute programs at the locations of the corresponding events. The programmer is isolated from the details of how these hosts and ports are implemented. When an external event (e.g., a vehicle) moves, the corresponding virtual host migrates with it transparently to the programmer.

This virtual host associated with an external event of interest is called an *entity*. Sensor nodes that can sense the event are called *entity members*. Members elect an *entity leader* that uniquely represents the entity and manages its state. Thus, an entity appears indivisible to the rest of the network. The fact that it is composed of multiple nodes with a changing membership is abstracted away.

When the external event moves outside the sensing horizon of the current entity leader, the leader hands off leadership to another member. Connection state is handed off as well, allowing communication with the entity to remain uninterrupted. To ensure unique representation of external events within the computational environment, a unique entity must be associated with each event. The transport protocol meets this constraint by announcing the existence of the entity to nearby nodes that cannot yet sense

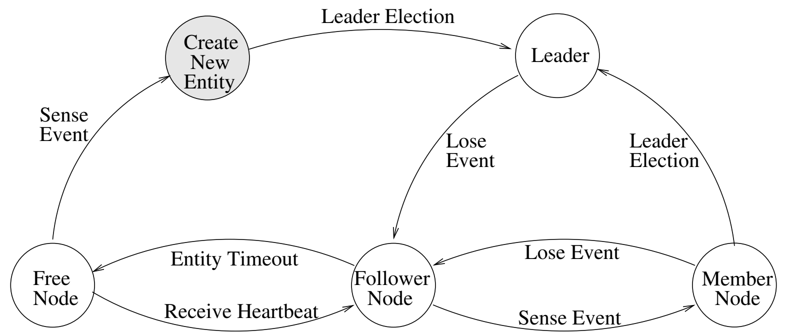

FIGURE 17.1 Node state transition.

the event. These announcements are sent periodically by the entity leader and are called *heartbeats*. Nodes that hear a heartbeat but cannot sense the event are called *entity followers* and are said to be within the *awareness horizon* of the named entity. Upon receiving a heartbeat, such nodes set an *entity timeout timer*; upon timer expiration, their status as followers expires. The timer is reset to zero every time a new heartbeat is received.

When the event enters the sensing horizon of a follower node, the node becomes a member of the entity it is following. If the node is not a follower, it recognizes that a new entity must be created. The node sets a random timer, upon expiration of which it claims leadership of the new entity. If it receives a leadership claim message from another node prior to timer expiration, it clears the timer and becomes an entity member. The algorithm ensures that a newly sensed event is represented by a single entity and that current events do not spawn spurious entities as they move from one location to another. Figure 17.1 depicts the node state transition diagram among follower, member, and leader states, as well as the free state in which a node is not cognizant of any entities.

An evaluation of this architecture reveals that entity uniqueness is maintained as long as the target event moves in the environment at a speed slower than half the nodes' communication radius per second [7]. For example, if sensor nodes can communicate within a 200-m radius, the transport layer can correctly maintain endpoints attached to targets that move as fast as 100 m/s (i.e., 360 km/h). The combination of this transport layer and the guaranteed velocity protocols described earlier provides invaluable support to real-time applications. For example, communication regarding moving targets can be made to proceed in the network at a velocity that depends on target velocity. Thus, positions of faster targets, for example, can be reported more quickly than those of slower ones. To the authors' knowledge, no other protocols in sensor networks have explicitly addressed message timing constraints.

17.3 A Sensor-Network Programming Model

The transport layer described previously gives rise to a programming model that elevates tracked activities in the physical environment into first-class programming abstractions. In this model, the application developer specifies events to be monitored. The system automatically detects such events and instantiates a so-called *context* every time an instance of an event is detected in the environment. From the programmer's perspective, the application is composed of a dynamic set of contexts, each representing a particular event. Objects can be attached to contexts and objects will logically execute in the locale of the monitored event. Contexts have unique identifiers called *context labels*. Objects attached to a context can be addressed using the context label and object name. They can communicate remotely by remote method invocation. The programmer's view of the application is depicted in Figure 17.2.

A context label around some event, e, is completely defined by two elements: (1) the function $sense_e()$, which specifies an environmental condition that spawns the context label; and (2) the function $state_e()$, which describes the environmental state to be encapsulated in the context label. The former function,

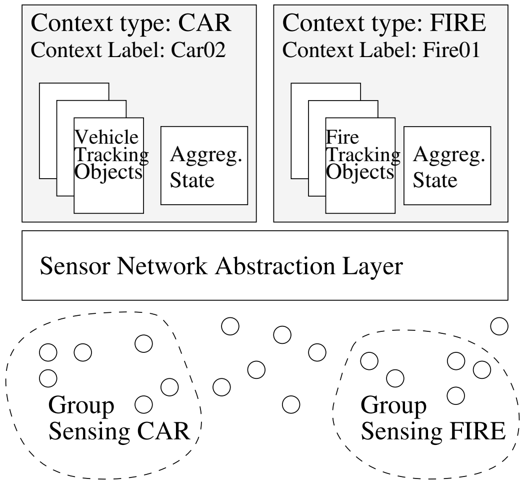

FIGURE 17.2 Programming model.

for example, might dictate that a label is to be created if magnetic distortion (e.g., the presence of a vehicle) is sensed. The state function returns a set of aggregate variables, each computed using outputs of at least N_e nodes for which *sense*$_e$() was true in the last L_e time units.

N_e and L_e are called the critical mass and freshness constraints, respectively. For example, to obtain the approximate position of a vehicle we may define *state*$_e$() may be defined to be the average coordinates of at least five nodes that have sensed the vehicle within the last 2 sec. Environmental tracking of event *e* is defined as the process of maintaining the state of this event subject to given freshness and critical mass constraints.

Syntactically, an application consists of a list of context declarations, each specifying an activation condition *sense*$_e$(); a set of state variables *state*$_e$(); and a list of attached objects. An example declaration is shown in Figure 17.3. The example defines a context of type *tracker* and specifies its activation condition, *sense*$_e$(), as an appropriate magnetometer reading (presumably caused by a nearby vehicle); it defines *state*$_e$() as the average *location* of the tracked target. It specifies that *location* must represent the average of at least two sensor readings measured no earlier than 1 sec ago.

The attached object is invoked periodically to report the current location of the vehicle to a virtual base station object. It passes the originating context label as the identity of the reported vehicle. If several vehicles are in the field, multiple reporter objects will be automatically instantiated. The programmer does not need to worry about instantiating these objects; object execution and maintenance of aggregate state occur automatically. Because details of the underlying communication, group membership management, leader handoff, and mobility are handled transparently, the programmer's interaction with the sensor network is significantly simplified.

This subsection has described real-time communication protocols and programming abstractions motivated by a tighter interaction between sensor networks and their physical environment. This architecture might be a first step toward a comprehensive vision for next-generation programming systems supporting future real-time deeply embedded distributed sensor network applications.

17.4 Related Work

Classical distributed programming paradigms and middleware such as CORBA [27]; group communication (e.g., ISIS [5]); remote procedure calls (RPC [6]); and distributed shared memory (e.g., MUNIN [9]) share in common the fact that their programming abstractions exist in a logical space that does not represent or interact with objects and activities in the physical world. Their main goal is to abstract distributed communication rather than facilitate distributed sensory interactions with an external

```
begin context tracker
  activation: MAGNETOMETER == ON
  location : avg (position) mass=2, freshness=1s
  begin object reporter
    invocation: PERIOD(0.5s)
    report_function() {
      BaseStation.reportLocation (self.label, location);
    }
  end
end context
```

FIGURE 17.3 Sample code.

physical environment. In contrast, sensor network applications call for a paradigm that revolves around "environmentally inspired" abstractions aimed at simplifying the coding of interactions with the physical world that arise in distributed deeply embedded systems.

The work reported in this chapter is closely related to several recent projects, such as Cricket [23], Sentient Computing [2], and Cooltown [10], that propose high-level paradigms in which an embedded distributed computing system is able to share humans' perceptions of the physical world. These systems allow the location of entities in the external environment to be tracked. One major difference is that they assume cooperative users who, for example, can wear beaconing devices that interact with location services in the infrastructure for purposes of localization and tracking [2, 23]. The authors' interest, in contrast, concerns situations in which no cooperation is assumed from the tracked entity.

In the absence of cooperation, several research efforts have proposed alternative addressing schemes that do not rely on having destinations with specific identities, but rather contact sensor nodes in the vicinity of a phenomenon of interest based on the attributes of data they sense. For example, DataSpace [17] exports abstractions of physical volumes addressable by their locations. Similarly, directed diffusion [14, 18] and the intentional naming system [3] provide addressing and routing based on data interests [14, 18]. Attributed-based naming is also related to the notion of content-addressable networks [24] proposed for an Internet environment, which allow queries to be routed depending on the requested content rather than on the identity of the target machine. Context labels, a form of attribute-based naming, are adopted. In this architecture, however, context labels are *active* elements. Not only do they provide a mechanism for *addressing* nodes that sense specific environmental conditions, but also they can *host context-specific computation* that tracks a target in the environment.

Recent research on system software for sensor networks has seen the introduction of distributed virtual machines designed to provide convenient high-level abstractions to application programmers, while implementing low-level distributed protocols transparently in an efficient manner [26]. This approach is taken in MagnetOS [12], which exports the illusion of a single Java virtual machine on top of a distributed sensor network. The application programmer writes a single Java program; the run-time system is responsible for code partitioning, placement, and automatic migration so that total energy consumption is minimized. Maté [20] is another example of a virtual machine developed for sensor networks. It implements its own bytecode interpreter, built on top of TinyOS [16].

A somewhat different approach to providing high-level programming abstractions is to view the sensor network as a distributed database in which sensors produce series of data values and signal processing functions generate abstract data types. The database management engine replaces the virtual machine in that it accepts a query language that allows applications to perform arbitrarily complex monitoring functions. This approach is implemented in the COUGAR sensor network database [8]. A middleware implementation of the same general abstraction is also found in SINA [25], a sensor information networking architecture that abstracts the sensor network into a collection of distributed objects.

This system is different in that it is geared for real-time environmental tracking. To the authors' knowledge, the first programming language for sensor networks that explicitly facilitates the coding of tracking applications and the first sensor network communication protocols that consider real-time constraints are described. These novel abstractions and underlying mechanisms are well suited for monitoring targets that move in the physical world. They can therefore have a major impact on application development for sensor networks.

17.5 Conclusions

This chapter reviewed a new protocol suite and programming system for sensor network applications that may considerably improve real-time behavior and reduce the development cost of deeply embedded systems. This reduction comes from off-loading the details of managing low-level abstractions from the application developer.

Future work of the authors will involve refinement of the real-time protocols and the environmental tracking problem so that more precise semantics and failure models are achieved. It is the hope that, with such refinements, a predictable sensor network "virtual machine" can be built that exports timely, reliable behavior and well-defined semantics, implemented on the unreliable, unpredictable, and resource-constrained hardware and communication infrastructure typical of sensor networks. Such a virtual machine would hide the complexity of sensor network programming from the application developer; it would make possible a new, more robust and dynamic realm of sensor network applications to affect future defense, surveillance, habitat monitoring, and disaster management systems.

References

1. I. Aad and C. Castelluccia. Differentiation mechanisms for IEEE 802.11. In *IEEE Infocom*, Anchorage, AK, April 2001.
2. M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, and A. Hopper. Implementing a sentient computing system. *IEEE Computer*, 34(8):50–56, August 2001.
3. W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *ACM Symp. Operating Syst. Principles*, Kiawah Island, SC, December 1999.
4. S. Basagni, I. Chlamtac, V.R. Syrotiuk, and B.A. Woodward. A distance routing effect algorithm for mobility (dream). In *ACM MOBICOM*, 76–84, October 1998.
5. K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Trans. Computer Syst.*, 9(3):272–314, August 1991.
6. A. Birrel and B. Nelson. Implementing remote procedure calls. *ACM Trans. Computer Syst.*, 2(1), February 1984.
7. B. Blum, P. Nagaraddi, A. Wood, T. Abdelzaher, J. Stankovic, and S. Son. An entity maintenance and connection service for sensor networks. In *1st Int. Conf. Mobile Syst., Applications, Services (MobiSys)*, San Francisco, CA, May 2003.
8. P. Bonnet, J. Gehrke, and P. Seshardi. Towards sensor database systems. In *2nd Int. Conf. Mobile Data Manage.*, 3–14, Hong Kong, January 2001.
9. J. Carter, J. Bennet, and W. Zwaenepoel. Implementation and performance of munin. In *ACM Symp. Operating Syst. Principles*, 151–164, October 1991.
10. P. Debaty and D. Caswell. Uniform Web presence architecture for people, places, and things. *IEEE Personal Commun.*, 8(4):46–51, August 2001.
11. D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: mobile networking for smart dust. In *ACM MOBICOM*, Seattle, WA, August 1999.
12. R.B. et al. On the need for system-level support for ad hoc and sensor networks. *Operating Syst. Rev.*, 36(2):1–5, April 2002.
13. T. He, J. Stankovic, C. Lu, and T. Abdelzaher. Speed: a stateless protocol for real-time communication in sensor networks. In *Int. Conf. Distributed Computing Syst.*, Providence, RI, May 2003.

14. J. Heideman, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. *Operating Syst. Rev.*, 35(5):146–159, December 2001.
15. J. Hightower and G. Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34(8), August 2001.
16. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. In *ASPLOS*, Cambridge, MA, November 2000.
17. T. Imielinski and S. Goel. Dataspace — querying and monitoring deeply networked collections in physical space. *IEEE Personal Commun.*, 7(5):4–9, October 2000.
18. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *ACM MOBICOM*, Boston, MA, August 2000.
19. Y.-B. Ko and V. Nitin. Location-aided routing (LAR) in mobile ad hoc networks. In *ACM MOBICOM*, 66–75, October 1998.
20. P. Levis and D. Culler. Mate: a tiny virtual machine for sensor networks. In *ASPLOS*, San Jose, CA, October 2002.
21. J. Li, J. Jannotti, D.D. Couto, D. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *ACM MOBICOM*, Boston, MA, August 2000.
22. C. Lu, B. Blum, T. Abdelzaher, J. Stankovic, and T. He. Rap: A real-time communication architecture for large-scale wireless sensor networks. In *Real-Time Technol. Applications Symp.*, San Jose, CA, September 2002.
23. N.B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *ACM MOBICOM*, Boston, MA, August 2000.
24. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Sigcomm*, San Diego, CA, August 2001.
25. C.-C. Shen, C. Srisathapornphat, and C. Jaikeo. Sensor information networking architecture and applications. *IEEE Personal Commun.*, 8(4):52–59, August 2001.
26. E. Sirer, R. Grimm, A. Gregory, and B. Bershad. Design and implementation of a distributed virtual machine for networked computers. In *ACM Symp. Operating Syst. Principles*, 202–216, Kiawah Island, SC, December 1999.
27. S. Vinoski. Corba: integrating diverse applications within distributed heterogeneous environments. *IEEE Commun. Mag.*, 14(2), February 1997.