An Efficient Protocol for Service Discovery in Wireless Sensor Networks

Mandeep Kaur Shilpa Bhatt Loren Schwiebert
Department of Computer Science
Wayne State University
Detroit, MI 48202
{mkaur, shilpa, loren}@wayne.edu

Golden G. Richard III
Computer Science Department
University of New Orleans
New Orleans, LA 70148
golden@cs.uno.edu

Abstract—**Service discovery is a widely researched topic in wireless networks. Existing protocols have significant overhead for service advertisement as well as service discovery, with many messages being transmitted in the network. This high overhead is not practical in resource-constrained wireless environments. We propose a simple service discovery protocol that aims at reducing the number of messages in the network while at the same time, keeping the waiting time for services to a minimum. We evaluate the performance of the protocol for a stationary wireless sensor network. We use remote procedure calls to request a service once it is discovered, which allows for richer and parameterized interaction with the sensor.**

I. Introduction

Wireless Sensor Networks (WSN) are a group of nodes, distributed in an area, that communicate with each other to collect information about the environment. The nodes are deployed in large numbers, usually hundreds or thousands, and can either have a fixed location or are randomly deployed [5]. They sense environmental changes and report them to other nodes over the network. These have been deployed for a variety of applications including habitat monitoring, medical monitoring, military surveillance, etc. [1], [2]. However, WSNs bring with them a host of challenges. Unpredictable wireless links, power constraints, memory constraints, and loss of individual nodes are only a few of these challenges.

As sensor networks become more popular, new applications that were previously unthought of are being developed. A flexible architecture that supports interaction among sensors that were originally deployed for different purposes will facilitate future uses of sensors.

Sensor nodes collect information from other nodes. For a dedicated application, this can be hard-coded into the application. But to allow new applications to be run on these sensor nodes, either from unanticipated interactions among overlapping sensor networks or by the arrival of additional wireless devices, some mechanism for finding sensors that provide information is required. Traditionally, this is known as service discovery.

Service discovery protocols are network protocols that allow automatic detection of devices and services offered by these devices on a computer network. Clients, the nodes that need a service, perform a discovery step, which typically initiates (limited) flooding of the network to discover nodes offering appropriate services. In some cases, clients may directly seek the needed services themselves; in others, they may contact one or more service catalogs, which maintain directories of available services. A discovery attempt generally classifies the service by type and may optionally include requirements such as a manufacturer, serial number, or other service attributes. Once the service providing node is found, it is important to ensure that the nodes can use the services efficiently. In this paper, we describe our protocol for service discovery in wireless sensor networks, TinySDP, and our initial implementation on TOSSIM, the simulator for TinyOS. An RPC [21] approach for requesting a service is also presented. This provides a more sophisticated interaction among the nodes.

II. Related Work

Jini is a service discovery technology based on Java [3]. In Jini, lookup services provide catalogs of available services to clients. Upon initialization, Jini services register their availability by uploading proxy objects to one or more of these lookup services. Once a client has contacted a lookup service, it can search for interesting services and then download the corresponding service proxy objects. Execution of methods in a proxy object allows communication with the corresponding service. However, Jini's dependability on Java makes it unsuitable for sensor networks.

SSDP provides a mechanism for HTTP clients and HTTP resources to discover each other in a local area network. It is used in Microsoft's UPnP (Universal Plug and Play) [4] architecture. When a service first comes online, it announces its presence in the network by sending a multicast message. All clients that hear this multicast cache the information. Any client that comes online after this announcement can discover the desired service by sending out a discovery request. As a result, messages are sent only when an event occurs. This event-driven approach, however, makes the protocol more complex. SSDP is based on HTTP, with large headers present in messages. Transmission of such large messages is inappropriate in sensor networks.

DEAPspace [18] is a decentralized discovery algorithm targeted at wireless ad-hoc single-hop networks. It uses a pure push-based approach for service discovery. The time is slotted into intervals and service information is broadcast during these intervals. The broadcasts are single-hop and are not transmitted beyond the local transmission range of the device in question. This is not suitable for general wireless sensor networks.

Gossip Based Discovery [17] is a service discovery protocol for mobile ad-hoc networks. A device collects information about services in the network by listening to all the service advertisement, request, and response messages in the network. A service registry on each device stores local services as well as services offered by others.

PDP [16] concentrates on service discovery in ad-hoc networks, where mobile devices communicate via wireless links without any fixed infrastructure. It is a fully distributed protocol that uses both push and pull techniques. In PDP, a device announces its services only when other devices request the services. The services in the PDP messages are defined using a URL scheme similar to the one used in the Service Location Protocol (SLP) [19] and transmitted using UDP or TCP. This introduces too much overhead to implement these protocols on wireless sensor nodes.

In the post-query model, each server posts its service(s) to a set of nodes according to the posting protocol and each client queries its desired services according to the querying protocol. A pair of Posting and Querying protocols forms a post-query protocol. In order to adapt to topological changes over time in an ad hoc network, these protocols are executed in rounds. Barbeau and Kranakis [14] propose various post-query strategies. These include:

- greedy; all nodes post and query all other nodes.
- incremental; all servers and clients post to and query. A small set of nodes in the first round and increase the size of these sets in subsequent rounds.
- uniform memoryless; servers post to a random set of nodes and clients query a random set of all nodes.
- uniform with memory; in each round a new set of nodes are posted to or queried.

Even though the above strategies work well and result in nodes successfully finding the services they need, all of them tend to consume high bandwidth. Multiple rounds in each strategy tend to consume a lot of resources in terms of power, bandwidth of the network, and memory in individual nodes.

May et al. present the design and implementation of a Remote Procedural Call (RPC) abstraction for nesC and TinyOS [22]. They present a programming abstraction to hide the complexities of inter-node communication both within and across a single-hop neighborhood. To achieve this they developed a set of nesC language extensions, a set of designer tools, and an operating system service for TinyOS. For discovery and binding, the designer replaces the destination node's ID with a special constant. The binding is persistent and future requests are sent to the same node. The proposed design is generic to wireless sensor networks; it does not cater to a lot of issues a service discovery protocol may have. Persistent bindings are used, which may not be ideal for wireless sensor networks.

Marionette [20] is a tool suite released by Whitehouse et al. using RPC for interactive development and debugging of wireless embedded networks. Using the Marionette architecture, embedded applications seamlessly span the PC and the sensor node. This is a tool for software development, not for run-time collaboration among sensor nodes.

Though many service discovery protocols have been proposed, these protocols are either more suitable for mobile ad-hoc networks (MANETs) or consume a lot of resources. The existing RPC based protocols either do not support inter-node communication or do not take into account all the requirements of a service discovery protocol. Instead of attempting to adapt an existing protocol, we have started with a basic list of needed functionality and, with the resource limitations of sensor nodes in mind, created a custom, extensible protocol for service discovery in sensor networks. We have also included an RPC component which allows a parameterized and more expressive communication among the nodes.

III. Application Scenario

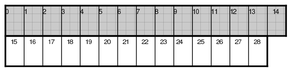
To illustrate the importance of service discovery protocols for wireless sensors, consider a network of nodes deployed in a forest to gather environmental information. A new wireless device enters this network to collect information. The types of information that the device is interested in collecting could differ based on the interests of the user. It could look for anything, varying from the number of sightings of a particular animal, the highest temperature recorded so far, or the humidity level in a particular area, etc. The flexibility to look for any desired service comes from standardization of the service discovery framework. This potentially saves the developers a significant amount of time over developing dedicated client/server systems for each application.

As another example, consider a sensor network deployed in a building. The thermal sensors can be accessed to provide consistent temperature control to determine hot spots during a fire. This information could be relayed to firefighters to aid in extinguishing the fire, and to occupants to guide them out of the building or to safer locations within the building.

For some applications, there are circumstances where a little more ”expressiveness” is required when communicating with a discovered sensor. For instance, consider the above example of nodes deployed in a forest. A user may need the temperature for a specific time, e.g., 3 am on March 11, 2008 or might be interested in changing the interval at which specific data is provided. An RPC-based service discovery protocol makes it easy to set parameters for such specific information.

IV. Tiny SDP

The Tiny Service Discovery Protocol defines a packet size of 29 bytes. We have a 15-byte application header and 14 bytes of data in our packet. As depicted in Figure 1, the protocol header carries the basic information about the packet. The packet type specifies whether the packet is an advertisement, request, reply, or acknowledgement. With an 8-bit field, we can define 256 different packet types. The 16-bit service type field specifies the service whose information the packet is carrying. However, some standardization of the service values is required to ensure that a particular value means the same service to all the nodes. For instance, if we define value 310 for temperature then this value will remain the same for every application. Some of these values are for defining standardized service types, leaving the rest for application-specific services. For generic message exchanges not concerning any specific service, the protocol requires a service type of zero.



• Byte 0:	TinySDP Version
• Byte 1:	Packet Type
• Bytes 2-5:	Source Node
• Bytes 6-9:	Destination Node
• Bytes 10-11:	Service Type
• Bytes 12-13:	Scope (0 == all scopes)
• Byte 14:	TTL (max hop count)

Fig. 1. TinySDP Packet Format

Every packet has a hop count (the time-to-live field). When hop count reaches zero, the packet is discarded. Limiting the lifetime of a packet using the hop count prevents old packets from moving around in the network.

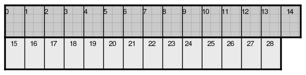
Services are grouped together using scopes. A scope is a set of sensors making up an administrative group. It can indicate a group of nodes within a particular location, within an administrative domain, or grouped by some other category. For instance, we may want to control the rights of nodes by allowing them access to services on some nodes and not on others. If we want a node to look for services in all domains, we use a scope of zero.

A. Service Advertisement

Sensor nodes advertise their services at network startup or when a new node enters an already existing network. The nodes can also periodically retransmit their advertisements. The source node initiates the advertisement. It fills in all the information in the packet including the service type that it is advertising. It also includes the current time and the time of service expiration. This expiry time is usually wired into the nodes when they are introduced into the network. This is the time duration for which the node will stay in the network.

B. Service Request

A node that needs to find a service in a network sends out a service request packet. This packet includes service requirements, time constraints, or other QoS parameters. As shown in Figure 2, the packet includes requirements about how long it needs a service, and with what frequency. In addition, we include ten bytes at the end of the packet for any special requests. For instance, consider a sensor network deployed in a factory to monitor industrial processes. A special request can look for a sensor that measures the amount of a certain harmful chemical or byproduct. In another case, a node monitoring temperature changes in a forest might send a request checking if the temperature has exceeded a certain threshold (indicating the possibility of a fire). This inclusion of special requests can help further reduce information exchange, thus controlling unnecessary traffic in the network.

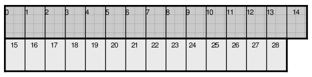


• Bytes 0-14:	Standard TinySDP header
• Bytes 15-16:	Time Duration
• Bytes 17-18:	Service Frequency requested
• Bytes 19-28:	Service-specific constraints

Fig. 2. TinySDP service request

C. Service Reply

When a node receives a service request, it checks if it meets the service requirements of the requesting node. If it meets the stated requirements, it will reply. The most important attribute values for the service (as defined in the standardization process for a service type) are piggybacked on the service reply (see Figure 3). This piggybacking of attributes is particularly useful when a node needs a one-time value for a service and potentially saves a significant amount of communication, by eliminating the need to perform attribute requests or RPC calls.



• Bytes 0-14:	Standard TinySDP header
• Bytes 15-18:	Current Time
• Bytes 19-22:	Service Expiration
• Bytes 23-24:	Service Frequency Available
• Bytes 25-28:	Piggybacked Standard Attributes

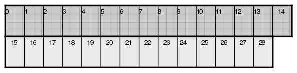
Fig. 3. TinySDP service reply

D. Service Acknowledgement

TinySDP provides an acknowledgement packet to allow retransmissions when necessary. If a request packet reaches its destination without finding the service, the destination node sends back a negative acknowledgement to the source node. Upon receiving this negative acknowledgement, if the source node hasn't already received a reply from another node, it retransmits in another direction.

E. Attribute Request/Reply

TinySDP supports two types of interactions with services – attribute exchange and RPC. To request attribute values, the client sends an attribute request. A service may maintain up to 32 bytes of attribute data. In the request packet, a 4 byte bitmask defines which of the 32 attribute bytes should be returned in one or more attribute replies. What attributes are specified by which bits is part of the standardization of the service. Also, the format of attribute data (e.g., whether it is an integer, a short character string, or a floating point value) is defined in the standardization process for a service. When a node receives an attribute request, it uses the attribute bitmask to pack the requested values into one or more attribute reply messages. A maximum of 10 bytes of data can be returned in a single attribute reply message, so an attribute request may result in more than one attribute reply. The attribute bitmask in the reply indicates which bytes of attribute data are being returned in that message (see Figure 4).

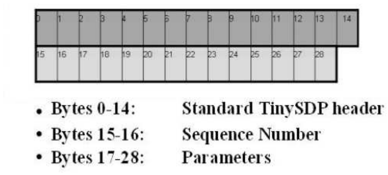


• Bytes 0-14:	Standard TinySDP header
• Bytes 15-18:	Attribute Bitmask
• Bytes 19-28:	Attribute Values

Fig. 4. TinySDP attribute reply

F. TinyRPC

TinySDP also provides an ultra-lightweight RPC mechanism, called TinyRPC, for complex client/service communication. RPC allows a client to initiate changes to a service node, for example, to request a change of scale for temperature readings or to request that readings be taken more often. The RPC mechanism also allows a service to maintain more than 32 bytes of attribute data. The packet type field in the standard TinySDP header is used to control RPC. Packet types greater than 128 correspond to service type-specific RPC calls, and are standardized for each service type. The RPC request and reply packet formats are identical, carrying the standard TinySDP header, a sequence number for matching RPC requests with replies, and up to 12 bytes of parameter data. After getting a service reply from a service-providing node and analyzing the piggybacked attribute values, the client may initiate an RPC request to get more specific information from the sensor.



• Bytes 0-14:	Standard TinySDP header
• Bytes 15-16:	Sequence Number
• Bytes 17-28:	Parameters

Fig. 5. TinyRPC service request/reply

V. Implementation details

We have developed a prototype implementation of TinySDP. We implemented our protocol using TinyOS version 1.1.11 and NesC, developed at UC Berkeley, for programming the motes. TinyOS provides an ad-hoc routing component architecture. We built our protocol on top of this component by defining a uniform packet format with a packet header for TinySDP. TinyOS follows an Active Messaging (AM) model, which means that every packet is associated with a handler ID that invokes a specific event on the receiver. The AM model also limits our packet size to a maximum of 29 bytes, which we took into account during initial development of the TinySDP protocol. The simulations were done in TOSSIM, which compiles directly from TinyOS code and runs the same code as on the Mica2 sensor hardware.

A. Service Discovery

In our implementation strategy we have tried to reduce the advertisement and discovery messages to a minimum. When a sensor node enters the sensor network, it distributes a service advertisement. This advertisement is sent across the network along a specific trajectory and cached by the sensors on the path. To find a service, a service request merely needs to intersect with a sensor along this path. The destination node field in the message can be used to carry the orientation of the probe. As long as the WSN has some reasonably accurate measure of location and uses some positioning system, even a local positioning system [6], then it is possible for sensors to find each other in the network.

We propose two strategies along these lines. First, service advertisements and service requests are sent along intersecting trajectories, as shown in figure 6. Sensor A, which possesses a service, advertises it along the North-South direction. Sensors on the path cache this advertisement and then forward the packet further. When sensor B needs a service, it sends a message in the East-West direction to intersect the vertical line. When it reaches a sensor that has cached the desired information, the sensor forwards the request to the service provider. However, if node B receives a negative acknowledgement, indicating that the service was not found, it can send another request in a different direction.

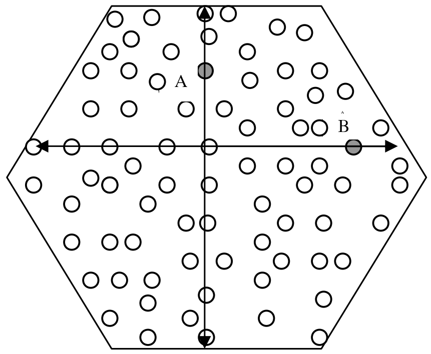


Fig. 6. Single Advertisement, multiple requests

A second, more robust idea, is to spread out the advertisement in the network. This idea is illustrated in figure 7. Sensor A, which possesses a service, sends out the advertisement in two directions, North-South and East-West. Sensor B, which needs a service, sends out multiple requests in directions at 45° angles relative to the advertisement. This guarantees that at least one message intersects the advertisement path. If the boundaries of the WSN are known, the orientation of service request messages can be calculated intelligently so that the request covers the maximum area of the network, reducing the number of request messages.

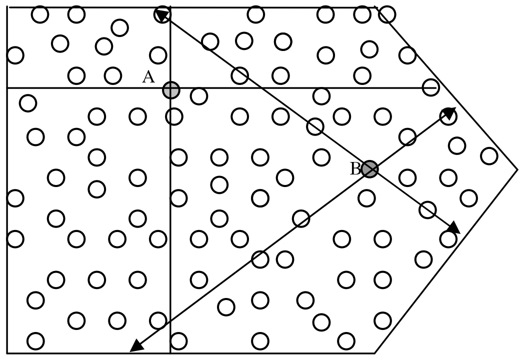


Fig. 7. Multiple Advertisement, multiple requests

B. Route Resolution

TinySDP requires a multi-hop routing protocol. We make a few assumptions while routing the packets. All nodes know their location with respect to a co-ordinate system [10] [12], the nodes are aware of their neighbors [8], they have a sense of directionality [9] and a source node can find the co-ordinates of the destination node from a location service [6] [7]. Since service discovery is a middleware protocol, we consider all these assumptions to be reasonable. This is the basic information needed to route packets in a network.

We use Trajectory Based Forwarding (TBF) as our routing protocol. It is a hybrid of source-based routing and cartesian forwarding [11]. Like source-based routing, the path is chosen by the source, but without specifying all the intermediate nodes. Like cartesian forwarding, decisions taken at each node are greedy, but are not based on distance to the destination – the measure is the distance along the desired trajectory. TBF removes the overhead at each node to maintain and update the routing information. Also, it trades off computation for communication. Considering the four orders of magnitude difference between the cost of sending a wireless packet and executing an instruction [13], TBF saves precious resources.

C. Data Structures

All nodes store some basic information, including information about a node's services, its neighbors, and advertisements received from other nodes. We have included only the most essential information in the cache entries. Every node uses three data structures for storing information. Two additional tables are required for use with RPC in service discovery: an RPC client table and an RPC server table.

1) Advertisement Table: The advertisement table contains the latest advertisements that the node has received. An advertisement is added to the advertisement table when it arrives. The expiration time is also saved. After the service expires, its entry in the table is removed or replaced with a new advertisement.

2) Service Table: The service table maintains the list of services that the node provides, including information like the time duration for which the service will be provided and the frequency with which it can be provided. Finally, the table stores some standard values for a service. For instance, it can store the maximum, minimum, or average value for the chemical concentration of an element in the past 48 hours.

3) Neighbor Table: This table lists all the neighbors of a node. The table contains a neighbor's address and its location. Before sending a packet, the next node is chosen from this table per the rules of the routing protocol. The format of the neighbor table may change according to the routing protocol used. For TBF, the node finds the neighbor towards the destination along a trajectory and then sends the packet.

4) RPC Client Table: The client table is maintained at the service-providing node and contains the service specific data related to the client it serves. The server, while processing the request, checks the client table and updates it if no entry for that client node is found. It records the service type it provides to the client and stores information for future responses to the client. This table is updated based on the service parameters in the RPC request. For example, the user may send a request to update the frequency at which it receives data.

5) RPC Server Table: The server table at the client node lists the address of each node providing it service. Along with the service type, it also records the scope of the service-providing node. This table stores the information of the service-providing node until the service is needed.

VI. Simulations and Results

A network with 50 nodes is simulated. All nodes are assumed to have the same resources in terms of power and memory. There are 11 kinds of services offered in the network. We assume that all services follow a uniform distribution. We use a random topology for the network. We have assumed a dense convex network with stationary nodes. To the best of our knowledge, no other protocols for service discovery on wireless sensor networks have been proposed, so we compare only our two proposed strategies.

The initial advertisements are sent when the network first starts up. Since we have 50 nodes in our network, if all of them start sending their advertisements together at network startup, a few of the advertisement packets are lost. In order to ensure that this doesn't happen, the nodes send their advertisement packets only in certain slotted time intervals. We give the network a time of two virtual minutes to start up before sending any request packets. The service requests are then randomly generated by the nodes in the network every three virtual minutes.

In order to analyze the results, we first define a few metrics. We denote the total number of nodes as N. The number of clients that successfully locate the service as N_{succ}. The number of service advertisement messages as M_{adv}, the number of service request messages as M_{req}, and the total number of messages sent as M_{total}. Each successful client c receives 1 to m service reply messages with waiting times of $t_1, t_2, ..., t_m$.

- Success ratio (SR): The ratio (as a percentage) of the number of nodes that successfully locate the service, over the total number of clients. It is calculated as:

$$SR = N_{succ}/N \times 100(\%) \tag{1}$$

- Number of transmitted messages (M$_{total}$): The total number of messages transmitted for the duration of the simulation. This is a useful parameter to estimate the efficiency of the protocol. It is calculated as:

$$M_{total} = \sum_{n=1}^{N} (M_{req} + M_{adv}) \tag{2}$$

- Average waiting time (AWT): The minimum time period in seconds, averaged over all the clients, starting from the transmission of a service request message and ending with the reception of a service reply message. It is calculated as:

$$AWT = \frac{\sum_{n=1}^{N} \min(t_1, t_2, ..., t_m)}{N} \tag{3}$$

TABLE I
Performance comparison

Strategy	RP	Max SR	M$_{total}$	AWT
TinySDP Strategy1	TBF	97.14%	323	0.425
TinySDP Strategy2	TBF	99.3%	541	0.59

Strategy 1 offers the least average waiting time and minimum number of messages in the network. The success ratio is comparable to Strategy 2. Strategy 2 offers a competitive average waiting time. However, the number of messages is more than Strategy 1. This is expected since we are sending multiple advertisement and service request messages.

Since both the TinySDP strategies have few messages, this saves network bandwidth and thus saves precious network resources. Low waiting time for discovering services is a measure of the high performance of our protocol.

The use of RPC does not require any additional inter-node communication beyond the requirements for the base service discovery protocol. RPC is built upon the service discovery protocol. Once the service-providing nodes are discovered, we can send RPC requests to these nodes. For RPC simulation, the client requests are generated in a random manner. Each simulation runs 50 different RPC requests generated from random nodes. We observed that when the protocol simulates 50 random RPC requests, the total number of packets transmitted, in a 50 nodes network, is 342. The number of packets transmitted depends on the hops a packet takes to reach the destination and the parameters in the service request. Since we know the destination of the service-providing node during an RPC call, the number of inter-node packets transmitted for service requests and replies will not change.

VII. Future Work

Some ways in which the protocol can be extended are:

- The direction of transmission of the messages could be computed intelligently based on the network information and location of the nodes.
- The results shown have been simulated on networks with convex boundaries. The work can be extended to concave networks. One of the ways in which this can be done is modifying the trajectories along which we send messages. TBF, the routing protocol used in our simulations, allows arbitrary trajectories. However, more research is required to find suitable trajectories for networks with concave boundaries, holes, obstacles, etc.
- We can also analyze the protocol with mobile nodes.
- Extending the RPC mechanism to highly dynamic environments depends on the efficiency of the underlying routing mechanism. This is a topic for further study.

References

[1] F. L. Lewis, "Wireless Sensor Networks," *Smart Environments: Technologies, Protocols, and Applications*, ed. D. J. Cook and S. K. Das, John Wiley, 2004.

[2] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Applications and OS: Wireless Sensor Networks for Habitat Monitoring," *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 88–97, 2002.

[3] J. Waldo, "The Jini architecture for network-centric computing," *Communications of the ACM*, 42(7):76–82, January 1999.

[4] Universal Plug and Play Specification, http://www.upnp.org.

[5] E. Stavrou, "Wireless Sensor Networks part 1: Introduction," 2005, http://webhosting.devshed.com/c/a/Web-Hosting-Articles/Wireless-Sensor-Networks-pt-1-Introduction/.

[6] D. Niculescu and B. Nath, "Localized positioning in ad-hoc networks," *Sensor Network Protocols and Applications*, pages 42–50, May 2003.

[7] J. Li, J. Jannotti, D. DeCouto, D. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," *Proceedings of ACM MobiCom*, August 2000.

[8] R. Madan and S. Lall, "An energy-optimal algorithm for neighbor discovery in wireless sensor networks," *Mobile Networks and Applications*, 11(3):317–326, June 2006.

[9] A. Nasipuri and K. Li, "A directionality based location discovery scheme for wireless sensor networks," *ACM International Workshop on Wireless Sensor Networks and Applications*, pages 105–111, 2002.

[10] D. Niculescu and B. Nath, "Ad hoc positioning system (APS)," *GLOBECOM*, pages 2926–2931, 2001.

[11] D. Niculescu and B. Nath, "Trajectory based forwarding and its applications" *Technical Report DCS-TR-488, Department of Computer Science, Rutgers University*, May 2002.

[12] D. Niculescu and B. Nath, "Ad hoc positioning system (APS) using AoA," *INFOCOM*, 2003.

[13] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister, "Smart Dust: Communicating with a cubic-millimeter computer," *Computer*, 34(1):44–51, January 2001.

[14] M. Barbeau and E. Kranakis, "Modeling and performance analysis of service discovery strategies in ad hoc networks," *International Conference on Wireless Networks*, pages 44–50, 2003.

[15] H. Luo and M. Barbeau, "Performance evaluation of service discovery strategies in ad hoc networks". In *Proceedings of the Second Annual Conference on Communication Networks and Services Research (CNSR 04)*, pages 61–68, 2004.

[16] C. Campo, C. Garcia-Rubio, A. M. Lopez, and F. Almenarez, "PDP: A lightweight discovery protocol for local-scope interactions in wireless ad hoc networks" *Computer Networks*, 50(17), 3264–3283, December 2006.

[17] C. Lee, S. Helal, and W. Lee, "Gossip-based service discovery in mobile ad hoc networks," *IEICE Transactions*, 89-B(9):2621–2624, September 2006.

[18] R. Hermann, D. Husemann, M. Moser, M. Nidd, C. Rohner, and A. Schade, "DEAPspace: Transient Ad-Hoc Networking of Pervasive Devices", *Computer Networks: Special Edition on Pervasive Computing*, 35(4):411–428, March 2001.

[19] E. Guttman, C. Perkins, and J. Kempf, "Service Templates and Service: Schemes," RFC 2609, Internet Engineering Task Force, 1999, http://www.ietf.org/rfc/rfc2609.txt.

[20] K. Whitehouse, K. Tolle, G. Taneja, J. Sharp, C. Kim, S. Jeong, J. Hui, J. Dutta, and D. Culler, "Marionette: using RPC for interactive development and debugging of wireless embedded networks," *The Fifth International Conference on Information Processing in Sensor Networks*, pages 416–423, April 2006.

[21] A. D. Birrell and B. J. Nelson, "Implementing Remote Procedure Calls" *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[22] T. D. May, S. H. Dunning, and J. O. Hallstrom, "An RPC design for wireless sensor networks," *IEEE International Conference on Mobile Adhoc and Sensor Systems*, 8 pages, November 2005.