Query Domains: Grouping Heterogeneous Sensors Based on Proximity

Vasanth Rajamani
Sanem Kabadayi
Christine Julien

TR-UTEDGE-2006-010

Query Domains: Grouping Heterogeneous Sensors Based on Proximity

Vasanth Rajamani, Sanem Kabadayı, and Christine Julien
The Center for Excellence in Distributed Global Environments
The Department of Electrical and Computer Engineering
The University of Texas at Austin
{vasanthrajamani, s.kabadayi, c.julien}@mail.utexas.edu

Abstract

Efficient query processing in sensor networks involves identifying groups of nodes that coordinate to satisfy applications' requests. In this paper, we propose a "query domain" abstraction that allows an application to dynamically specify the nodes best suited to answering a particular query. To self-organize into such a coalition, nodes must satisfy a "proximity function", a user-defined function that constrains the relative relationship among the group of nodes (e.g., based on a property of the network or physical environment or a logical property of the nodes). The proximity function removes the need to explicitly tag nodes with context information, and it provides a convenient mechanism for forming coalitions on-the-fly at query time. This facilitates the deployment of general-purpose sensor networks, where multiple applications can be run in the same network at the same time. In this paper, we model this abstraction, present a protocol to support the abstraction, and evaluate their performance.

1 Introduction

Miniaturization has enabled the production of inexpensive battery-operated sensors that contain one or more modules to sense various aspects of the physical environment, such as temperature, humidity, etc. When sensors deployed on a large scale, however, there is an explosion in the amount of data to observe and analyze. Just as the plethora of web data was largely human-unusable until the advent of modern search engines, querying techniques will play a pivotal role in comprehending sensor data.

Sensor networks have been deployed in a wide range of applications such as habitat monitoring [8], intelligent construction sites [3], and industrial sensing [5]. A common theme that emerges from these applications is the need for querying different types of data simultaneously to answer a particular question. An illustrative example is in intelligent farming. Fungi typically infect potato plants when the temperature drops below 10° Celsius and the relative humidity exceeds 0.90 [11]. A crop failure detection application needs to simultaneously monitor data from both temperature and humidity sensors, and when the conditions become unfavorable for the crop, an alert must be raised. As wireless sensors are deployed more regularly in such environments, it becomes imperative that query processing techniques efficiently handle heterogeneous data types.

With the sensors' decreasing sizes and increasing integration into the environment, the individual capabilities of each sensor are likely to become increasingly specialized. Therefore, multiple sensing devices must cooperate in the resolution of a single query. Practically, the sensors selected to work together should satisfy some application-specific constraints. Such relationships between sensors that contain the desired sensing modules cannot be known *a priori* and are unique for each application.

In this paper, we introduce a *proximity function* which allows an application to specify the constraints between the different sensors used to answer a single query. These constraints are injected into the network, and sensor nodes that can satisfy the query data type requirements *and* the proximity function's relationship constraints self-organize into a logical query domain. Only the small subset of sensors in the query domain participate in replying to the query. Once organized, the query domain becomes a convenient handle to repeatedly communicate with question of the same coalition of sensors minimizing energy consumption.

The novel contributions of this work are threefold. First, we define two new abstractions, the *query domain* and the *proximity function*, that allow an application developer to identify the dynamic set of sensors that answer a query by specifying constraints on the relationships between the responding sensors. The query domain is calculated in a dis-

1-4244-1455-5/07/$25.00

tributed fashion using the proximity function and allows a sensor network to be used for multiple applications, moving away from current deployments where a sensor network is tailored for a particular application. We devise a fully reactive approach that calculates and constructs the query domain completely on-demand, enabling the networked sensors to self-organize. Finally, we evaluate the effectiveness of this approach through simulation.

2 Motivation and Problem Definition

In this section, we concisely state the research problem we address and motivate the need for a solution by providing concrete examples where the stated issues materialize.

2.1 Problem Statement

This paper addresses the problem of dynamically defining a sensor coalition that collectively acts as a single entity in responding to an application's query. This must be done without any *a priori* knowledge, and therefore nodes must self-organize according to some application-specified rules provided at run time.

2.2 Application Scenarios

Crop Failure Prevention. Phytophtora is a fungal disease in crops that affects potatoes. Typically, the fungi sporulate when the temperature is low (less than 10°C) and the relative humidity is high (greater than 90%) [11]. While it is possible that both temperature and humidity sensors are available on the same device, it is also possible that that the network contains devices with more limited capabilities. A coalition of temperature and humidity sensors could provide the same functionality as the single more complex sensor. The query for detecting deteriorating Phytophtora conditions may be stated as: "Select temperature and relative humidity readings from sensors that are within 10 meters of one another." In this example, the user wishes to query temperature and relative humidity sensors, and the constraint he designates is that the two sensors be near (within 10 m of) one another.

Habitat Monitoring. Wireless sensor networks have been deployed in a variety of habitat monitoring environments [8]. A field biologist may be interested in studying the mating behavior of a species without human presence using a combination of audio and video sensors deployed in the field. The arrival of the animals may be detected by the audio sensors, which in turn trigger the video sensors to record and transmit images. It is critical that the latency between the audio and video sensors be within a stated threshold; a high latency can result in the reception of video images after the animal has left the area, leading to wasted video feeds. Also, the distance between the sensors must be small, or the audio sensors will detect the arrival of the animal, while the video sensors will be recording and transmitting images from a completely different location.

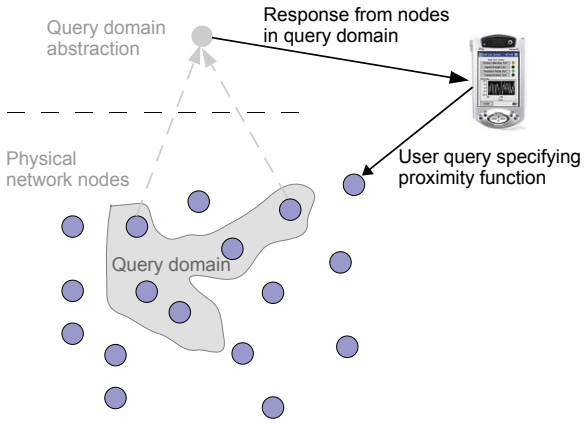

Figure 1. Query Domain

3 Abstraction

A query domain associates a query with a dedicated neighborhood that satisfies a user-defined *proximity function*. In contrast to existing approaches that group nodes based solely on their content, our approach groups nodes based on relationships as well as content. Proximity functions may include a variety of user-defined predicates like hop count, displacement, latency, or bandwidth criteria. It is important to note that the query domain and its associated proximity function define a relationship that all responders to the query must satisfy among themselves. Distributing the proximity function to nodes in a sensor network allows an application to dynamically impose an overlay structure on the network, impacting how the network behaves in response to the application's queries.

Figure 1 gives a pictorial depiction of a query domain. As shown, the members of the overlay are constrained by the application-specified proximity function. This domain is created by the application at query time and attached to the query to allow for distributed computation. Because a query carries with it the definition of its query domain, nodes in the sensor network are not required to know *a priori* what types of query domains the application will request, alleviating the need to tailor a sensor network to a particular application.

3.1 Formalization

There are two components required for formally specifying the behavior of our abstraction. The first of these states

the form of an application's query. The second formalizes the semantics of the query domain supporting this query. Section 4 describes the protocols we use to implement this abstraction. A query can be written as:

$$Q : \{t_1, t_2, \ldots, t_n\}/F_p$$

Any query is defined first by the data types required (i.e., $t_1, t_1, \ldots, t_n$) and second by the proximity function F_p applied to constrain the query domain. F_p is an application-specified Boolean function that takes any pair of nodes and returns true if the pair satisfies the proximity function and false otherwise. When referring to components of a query, we use the "dot" notation. That is, the set of types a query requires $(t_1, t_2, \ldots, t_n)$ can be retrieved using $Q.\text{types}$, while the proximity function can be retrieved using $Q.F_p$. The set of types any one sensor in the network can provide is referenced in a similar way, as $s.\text{types}$.

A query domain for a query, Q, is therefore any set, S, of nodes that satisfies the following four conditions[1]:

$\langle \forall s : s \in S :: s.\text{types} \cap Q.\text{types} \neq \emptyset \rangle \wedge$
$\langle \forall t : t \in Q.\text{types} :: \langle \exists s \in S :: t \in s.\text{types} \rangle \rangle \wedge$
$\langle \forall s : s \in S :: |(S - \{s\}).\text{types} \cap Q.\text{types}| < |Q.\text{types}| \rangle \wedge$
$\langle \forall s_1, s_2 : s_1, s_2 \in S :: Q.F_p(s_1, s_2) \rangle$

The first of these constraints requires that each sensor in the query domain S have at least one of the query's required types. The second condition requires every type requested in Q to be provided by at least one sensor in S. The third constraint ensures that none of the sensors selected for the query domain are redundant. That is, this condition guarantees that, if any sensor s is removed from S, then the types remaining in S are insufficient to satisfy Q. (In this condition, the notation $|S|$ refers to the cardinality of the set S.) The fourth and final condition ensures that the proximity function F_p is true for every pair of sensors. This also implicitly requires that F_p be reflexive, i.e. $F_p(s, s)$ is true.

3.2 Application Examples

Given the formalization of queries above, we next show how applications described in Section 2 can use our model and the proximity function abstraction.

Crop Failure Prevention. This query has the form:

$$Q : \{\text{temperature}, \text{humidity}\}/\text{distanceProximity}_{10m}$$

where the types within the braces are the two data types the application must find in the network, while distanceProximity refers to a (built-in) proximity function that requires all of the sensors in the query domain to be within a given distance of each other. For proximity functions that require such thresholds, the threshold (e.g., 10 meters) is given as a subscript. Formally, this indicates an infinite number of distanceProximity functions, one for each possible value of the threshold; this is implemented as a parameterized function. It is also possible that a proximity function requires more than one threshold; in such a case, the subscripts are separated by commas and refer to multiple parameters.

Habitat Monitoring. For the habitat monitoring application example, we require audio and video sensors that are connected by network paths that provide some maximum latency (for example, 10 seconds):

$$Q : \{\text{video}, \text{audio}\}/\text{latencyProximity}_{10s}$$

Applications may also require that the sensors in the query domain be physically close together to provide some confidence that the video sensor is capturing relevant video. To augment the above to account for distance in addition to latency, we simply apply a second cost function:

$$Q : \{\{\text{video}, \text{audio}\}/\text{latencyProximity}_{10s}\} /\text{distanceProximity}_{10m}$$

Logically, these two proximity functions are combined into one larger function with the cumulative constraints of each function. Alternatively, the fourth condition from the previous section could be rewritten as:

$$\langle \forall s_1, s_2, i : s_1, s_2 \in S \wedge i < m :: Q.F_{p_i}(s_1, s_2) \rangle$$

where m refers to the number of proximity functions applied to the query.

4 A Query Domain Assessment Protocol

In this section, we present a protocol that uses the proximity function to dynamically create query domains. Table 1 presents a summary of the abbreviations we will use in describing this protocol, some of which were introduced in Section 3.

The goal of a protocol for forming a query domain is to locate a set of nodes, S, that can satisfy a query specified as described in the previous section. The protocol should return to the query issuer a return path, P, that the querier can subsequently use to contact the constructed query domain.

Our completely reactive protocol for query domain construction sends an initial flood looking for a sensor that can satisfy at least one of the data type requirements (i.e., a sensor that supports one of the types in $Q.\text{types}$). When such a sensor is found, this sensor initiates a secondary flood looking for the remaining unsatisfied types from $Q.\text{types}$.

[1] In the three-part notation: $\langle \text{op}$ *quantified_variables* : *range* :: *expression*$\rangle$, the variables from *quantified_variables* take on all possible values permitted by *range*. Each instantiation of the variables is substituted in *expression*, producing a multi-set of values to which op is applied. If no instantiation of the variables satisfies *range*, then the value of the three-part expression is the identity element for op, e.g., *true* if op is $\forall$.

Table 1. Definitions

Query's Type Set ($Q.\text{types}$)	The set of sensor types required for this query
Proximity Function ($Q.F_p$)	The user-defined constraint on membership in the query domain
Path (P_{ij})	A series of links in the network that transitively connects nodes i and j
Return Path (P)	A connected graph of nodes selected for a query domain

This first flood is constrained only by a network time to live (TTL). The second flood can be constrained by the nature of F_p if the proximity function operates on network properties (e.g., hop count or latency). On the other hand, when the proximity function is defined by physical properties (e.g., distance) or logical properties (e.g., the same crane), the second flood is also constrained by the network's TTL. This allows our protocol to ensure the pairwise constraint detailed in the previous section while potentially limiting the overhead associated with constructing the query domain.

As these secondary floods are initiated, the query keeps track of the path it has traversed so far. When all of the types from $Q.\text{types}$ have been satisfied, the final sensor in the chain sends the return path P to the query initiator. This return message contains the values for each of the sensors in the query domain *and* the information for the query issuer to issue subsequent queries to the same domain.

Figure 2 shows this process, focusing on three cases a query domain resolution may encounter. In this analysis, we consider a query that requests only two data types; queries for more types of data incur subsequent constrained floods. The first case (the path involving node j) is our base case. In this case, node i was a hit for one of the data types in $Q.\text{types}$. After this match, node i initiated a secondary query, constrained by either the proximity function F_p or the TTL, as described above. Node j matches the second data type required by $Q.\text{types}$, and j sends the return path $j - i - p'$ back to the query issuer, where p' is the series of hops the query had to traverse before reaching node i.

For the second case, consider the path shown to node m. In this case, node i initially matched one of the data types of the query. Again, i initiated a secondary query, but k (and all the nodes between k and m) could not satisfy the remainder of the query. These nodes continued to forward the query until it landed at node m, which could provide the second data type. Node m returns to the query issuer the return path $m-p''-k-i-p'$, giving the query issuer access to the entire query domain, the set $\{i, m\}$. The return path itself is not the query domain, but it enables the query issuer to contact every device that is within the query domain.

In the final case, the path terminating at the node labeled o, node i also satisfied one of the two required data types for the query. However, in forwarding the query in search of the second required data type, either the proximity function was exceeded along the path (in the case that the proximity function is based on properties of the network topology) or the network's TTL was exceeded. In this case, no return path is sent back to the query issuer because no query domain was formed along this search route.

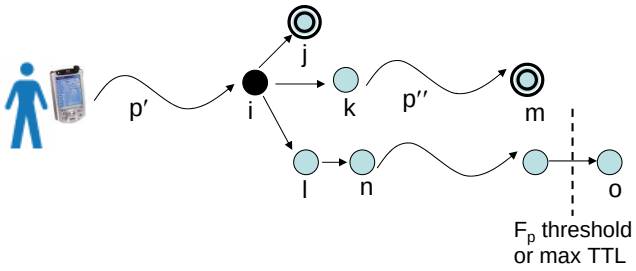

Figure 2. Reactive protocol details

5 Evaluation

In this section, we provide an evaluation of our protocol. To perform this evaluation, we used the TOSSIM network simulator [6], which allows direct simulation of TinyOS code written for MICA2 motes.

5.1 Simulation Setup

We used a uniform random placement of sensor nodes in a 100 x 100 foot area. We used TOSSIM's disc model with a radius of 10 feet to model radio transmissions. To distribute sensor types, we took all of the possible types for a particular simulation, created all possible combinations of those types, and randomly dispersed the combinations in the network. This is representative of real environments where powerful nodes that can support all data types are likely to be rarer than simpler devices that can provide partial support for a query. For example, when there are three possible sensor types, there are seven possible combinations of sensors (a node can have any one of the three available types, any two of the three types, or all three types).

5.2 Simulation Results

In our simulations, we varied the number of nodes from 50 to 150 in increments of 25. To study the feasibility of creating query domains using proximity functions, we implemented two types of query domains. The first proximity

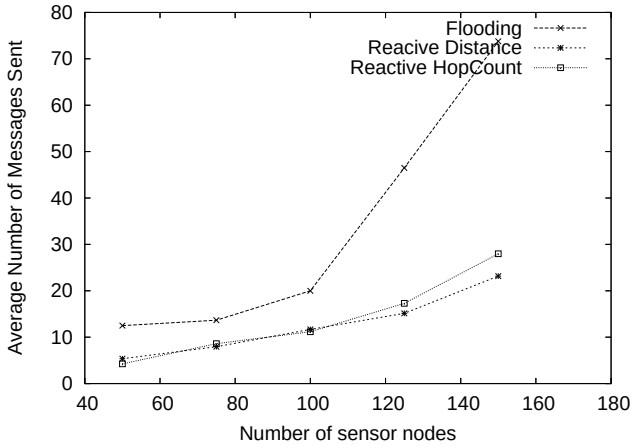

Figure 3. Number of messages transmitted per query

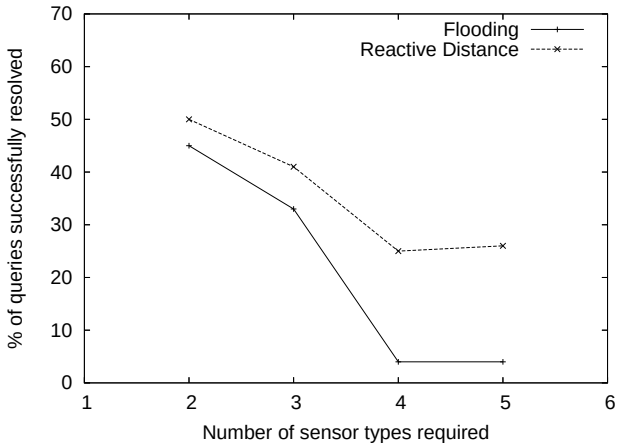

Figure 4. Percentage of successfully resolved queries

function ensures that all nodes be within four hops of one another. The second is a more sophisticated example where the proximity function stipulates that the distance between each member is no greater than 10 feet.

Figure 3 shows the number of messages transmitted by the sensors to resolve a query. The graph compares our protocol to flooding. The flooding protocol broadcasts the necessary sensor types to every node in range of the query issuer. Recipients of the broadcast reply if they have all the required sensors. Otherwise, the message is re-broadcast until the required sensors are found or the network's TTL is reached. The number of messages increases as the number of nodes increases because the node density increases, causing the number of nodes in re-broadcasting range to increase. We have simplified the flooding results to one line because the flooding implementations based on distance and hop count had very similar results. Regardless of the particular proximity function, our protocol outperforms the flooding-based protocol, especially as the number of nodes in the network increases. This is due to the fact that our protocol can use its hierarchical flooding mechanism to more carefully scope the query's dissemination, thereby reducing the overhead of discovery. In addition, the query domain construction protocol need not search as widely for a match since nodes can cooperate to resolve the query.

Figure 4 shows the percentage of queries that were successfully resolved as the queries became increasingly complex. This metric determines whether it is effective to rely on collaborative behavior as opposed to simply searching for powerful nodes. Henceforth, all our simulations will use the distance-based proximity function. To increase complexity, we varied the number of sensor types required by the query from two to five, while holding the number of nodes in the network constant at 100. As can be seen from Figure 4, as the number of sensing capabilities required for a single query increases, the probability of finding a satisfactory sensor or set of sensors drops substantially. In fact, as the number of sensors increases to about four or greater, the probability of finding one all-powerful node with all desired functionality drops to near zero. The query domain facilitates the collaboration of several less powerful nodes on complex tasks.

These results show that the query domain abstraction scales well as the number of nodes in the network increases. The communication overhead is favorable when compared to alternatives. As the complexity of sensor nodes increases, the query domain abstraction offers a higher percentage of successful query resolution.

6 Related work

Early work in grouping abstractions focused on creating groups in the physical neighborhood. Hood [14], Abstract Regions [13], and EnviroTrack [1] provide neighborhood abstractions with the goal of easily encapsulating membership, data sharing, and messaging between group members. These approaches define only physical neighborhoods. Additionally, they require proactive exchanges of attributes to form neighborhoods. In contrast, *logical neighborhoods* [9], focuses on constructing overlay networks defined by logical properties exported as tagged information. Our approach is less restrictive because we form the query domain reactively at query time. This reduces the amount of information that needs to be programmed in the network at deployment time, facilitating general-purpose sensor network deployments. While our proximity function can be

used to group nodes that have tagged information, it is most advantageous when it is used to create groups on the fly based on arbitrary relationships imposed by the application.

In [4], the authors focus on grouping abstractions in a pervasive environment. Here, a mobile device forms a *scene* around itself which allows applications to specify the types of information and aggregation operations on data within the scene. Scenes specify the relation the nodes must have to the user while our work focuses on the relationship nodes have to one another.

Our work also overlaps with systems that try and make complex query processing easier. ACQUIRE [12] provides a mechanism to resolve complex queries into a sequence of simple queries. Unlike our work, ACQUIRE does not form a group while processing complex queries. For persistent queries, our abstraction can be leveraged repeatedly, reducing energy usage in resource discovery, while ACQUIRE must re-establish which sensors to use every time. An alternate approach to query processing is to build high-level systems like TinyDB [7] and Cougar [15], which make the user completely oblivious to the underlying network. It is viewed as a relational database which can be queried using an SQL-style syntax. Our proximity function can be used to specify arbitrary functions based on physical properties, which is inconvenient to express in these systems.

The final category of related work includes systems that simplify programming of entire sensor networks. Examples of such macroprogramming systems include Kairos [2] and Regiment [10]. The primary goal of these systems is to try and automate the process of writing custom code for individual sensor nodes. Our work is complementary and our abstractions can be built into these systems.

7 Conclusions

We have described the query domain abstraction that enables the best-suited coalition of sensor nodes to answer a particular query. The specification of the proximity function to form the query domain reduces the communication overhead, since it does not require any communication *a priori*. This also makes it suitable for supporting multipurpose sensor networks. We have related the abstraction, its protocol implementation, and performance evaluation demonstrating the protocol's scalability with increasing number of nodes.

Acknowledgments

The authors would like to thank the Center for Excellence in Distributed Global Environments for providing research facilities and the collaborative environment. This research was funded, in part, by the National Science Foundation (NSF), Grants # CNS-0620245 and OCI-0636299. The conclusions herein are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

[1] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. Luo, S. Son, J. Stankovic, R. Stoleru, and A. Wood. EnviroTrack: Towards an environmental computing paradigm for distributed sensor networks. In *Proc. of ICDCS*, pages 582–589, 2004.

[2] R. Gummadi, O. Gnawali, and R. Govindan. Macroprogramming wireless sensor networks using Kairos. In *Proc. of the Int'l Conf. on Dist. Comp. in Sensor Sys.*, pages 126–140, 2005.

[3] J. Hammer, I. Hassan, C. Julien, S. Kabadayı W. O'Brien, and J. Trujillo. Dynamic decision support in direct-access sensor networks: A demonstration. In *Proc. of the 3^{rd} Int'l. Conf. on Mobile Ad-hoc and Sensor Systems*, 2006.

[4] S. Kabadayı and C. Julien. A local data abstraction and communication paradigm for pervasive computing. In *Proc. of PerCom*, March 2007.

[5] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *Proc. of SenSys*, pages 64–75, 2005.

[6] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proc. of SenSys*, pages 249–260, 2003.

[7] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: An acquisitional query processing system for sensor networks. *ACM Trans. on Database Systems*, 30(1):122–173, 2005.

[8] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. of the Int'l. Workshop on Wireless Sensor Networks and Applications*, pages 88–97, 2002.

[9] L. Mottola and G. Picco. Programming wireless sensor networks with logical neighborhoods. In *Proc. of InterSense*, 2006.

[10] R. Newton and M. Welsh. Region streams: functional macroprogramming for sensor networks. In *Proc. of the 1^{st} Int'l' Workshop on Data Management for Sensor Networks*, pages 78–87, 2004.

[11] Potato and Onion Disease Models. `http://www.metos.at/diseasemodels/Potato.pdf`, 2006.

[12] N. Sadagopan, B. Krishnamachari, and A. Helmy. Active query forwarding in sensor networks (ACQUIRE). *Elsevier Ad Hoc Networks*, 3(1):91–113, January 2005.

[13] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *Proc. of NSDI*, 2004.

[14] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: A neighborhood abstraction for sensor networks. In *Proc. of MobiSys*, pages 99–110, 2004.

[15] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(3):9–18, 2002.