

Scalable Decentralized Control for Sensor Networks via Distributed Lattices

Baruch Awerbuch¹ and Jonathan Stanton²

¹Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
baruch@cs.jhu.edu

²Department of Computer Science
George Washington University
Washington, DC 20052
jstanton@gwu.edu

Abstract. A network of embedded devices needs to be able to execute queries for dynamically changing content. Even in a completely reliable network, this is a formidable task because of the enormous scale of the networks, severely limited resources of individual devices (bandwidth and battery power) and the heterogeneity of resources being managed. In this work we introduce a novel information query methodology for designing online solutions for heterogeneous sensor networks with various resources (e.g., battery, bandwidth, CPU). This provides a route selection and query management mechanism that will enable a sensor network to find sensor level information without a routing algorithm specialized for the particular form of information. In order to execute such a methodology in a scalable, limited resource environment such as sensor networks we employ a novel lattice data structure, which is basically a combination of trees with small overlap that provably enables extension of any routing or directory infrastructure to an arbitrary scale, with only small overhead. We show how to use such data structures (lattices) that will enable scaling to millions of devices with an overhead that only grows logarithmically in the number of network nodes and with provably small distortion of paths. Moreover, we show a completely distributed implementation of such structures, that creates minimal overload on the client sensors.

1 Introduction

Content addressable routing, namely routing towards the location storing certain data is quite appropriate for embedded networks of many small devices. Instead of addressing a specific device, one needs to route a query to a device that has specific information, e.g. a sensor on (some) bridge. Notice, that content addressable routing greatly increases the scale of the problem (the number of queries is much larger than the number of nodes). In some way, since contents is variable, content addressable routing is equivalent to routing in a network where nodal names are changing arbitrarily (based on the contents).

The sheer scale of embedded and sensor networks appears to force a decision based on local information, since the overhead of obtaining the global information may overwhelm the benefit of using it. There is an analogy between routing, content retrieval, and maintenance of local databases of topology and resource depletion information. This analogy is expressed in trade-offs between different generic approaches.

Flooding each piece of information through the network is one extreme. This guarantees the highest quality of information and in a sense simulates global decision making, which results in efficient queries of content or routing of a message. However, the overhead of maintenance is prohibitive and overwhelms the benefits of obtaining the information.

Another extreme is on-demand DSR routing [5] which proceeds by flooding a routing (or information) query upon each request. While this is very expensive operation, it may be meaningful for content-addressable routing, e.g. “connect to the sensor near a human”, where there is simply too much content to keep track of.

The above arguments indicate an inherent trade-off between overhead in lookup and overhead in maintenance of data structures. This trade-off has been considered in a number of different papers. Sophisticated methods to examine various trade-offs in matching the sources of queries with sources of content such as directed diffusion [12] illustrate this approach. There is also work on aggregating data from different sources via distributed approximate set cover algorithms [11].

One attempt to avoid this trade-off is geographic “Location aided routing” [18] where the routing request is indicative of the position of the destination; e.g. its GPS coordinates; routing proceeds by making incremental steps toward the destination. This certainly does not work for dynamic content-addressable routing.

Note the assumption that addresses of routing destinations are bound to their geographic location, which inherently assumes that both naming and geography are static. The major (and not so obvious) fallacy of such an approach is the assumption that greedy progress in terms of geographic proximity to the destination is the right strategy. Such a strategy may work well assuming uniform distribution of devices in each geographic region, as well as completely uniform traffic distribution. However, it is easy to see that in fact traffic distribution is highly non-uniform (e.g. few producers and consumers of information). In this case, this strategy may deplete batteries of nodes along the geographically shortest paths, and shut these nodes off. This invalidates the “uniform geographic distribution” assumption, as well as the binding between names and geography leading to potentially catastrophic consequences, since geography is assumed to be static.

Attempts have been made to combine geography with battery utilization, such as GEAR [17] (geographical and energy aware routing); yet there is no rigorous mathematical argument for this strategy being close to optimal. Obviously, the geographic approach is not applicable in this case since the name has nothing to do with the underlying geography.

We address this tradeoff between content dissemination costs and query costs with a general routing algorithm and distributed data-structure that balances the costs while requiring very small storage at each node and small communication over each link.

In the rest of this paper we present this routing algorithm and its properties in several stages. First, in Section 2 we discuss the core idea in the context of a basic dis-

tributed tree. We then present the lattice based algorithm, that resolves the limitations of the tree-based one in Section 3. Section 4 discusses the related work and Section 5 presents our conclusions.

2 Tree-based Routing

Imagine that our communication network was a tree, and that the costs of communication over edges of the tree would increase exponentially with the distance from the leaves; we will call such a network a “fat tree”. Imagine that all the content is stored at the leaves of the tree, also called clients, and all the control information is stored at the internal nodes. The internal nodes may be separate embedded “server” devices which have additional memory and energy, or they may be a subset of the sensor devices. Since the internal nodes do work, by storing aggregated content and handling routing messages for other nodes they will use up resources faster than the leaf nodes (pure sensors). Thus, the actual devices acting as internal nodes will either need additional resources, or will have to be rotated in and out of providing those services to maximize the lifetime of the nodes. We will discuss later, when the distributed algorithm is presented how these trees can be dynamically modified and maintained in the context of these type of heterogeneous nodes.

2.1 Our Approach – Tree version

We can aggregate information and accomplish content addressable routing, as well as dissemination of resource utilization information on such fat tree as follows.

Content, originating at a leaf sensor, is registered at all levels of its tree ancestors. This is easily accomplished by climbing up the hierarchy. A query climbs up the hierarchy to locate the lowest common ancestor storing the content, and then descends down the hierarchy to the location of the contents.

Once this hierarchical structure has been created, it can also be used to aggregate the resource costs in a scalable way. The cost information is fairly simple data to aggregate because different costs can simply be added together to reach a combined cost. As long as all the costs have been calculated as opportunity costs, the sum of them, say from one cluster of nodes in the network, represents the aggregate opportunity cost of traversing that cluster of nodes.

Servers on level n can aggregate the resource costs reported by the level $n-1$ nodes (servers or sensors), and forward them further as a single value, usually the summation of the costs received. By doing this the cost updates only need to be sent locally among the sensors local level 1 cluster, then the controller (or server) of that cluster will incorporate that cost into the aggregate cluster cost and only send that cluster cost to other level 1 neighboring clusters and to the parent level 2 cluster leader. By using this aggregated cost information a controller at the n -th level of hierarchy can avoid a congested $n-1$ area as a whole, without knowing resource costs of each individual sensor.

2.2 Simple distributed implementation

We could try in principle to implement such a data structure in an embedded network. In order to do this we need to accomplish three tasks: decide which nodes should play a role at each level of the tree; build the tree; and specify the algorithm for routing within the tree.

For the first task of determining which nodes should participate at each level of the tree, we want to create the tree such that a nodes who are have a “small” communication cost between them are close together in the tree. We do not want to only consider physical or network distance because other factors such as bandwidth and battery life can substantially change the *effective* cost of traversing nodes – even if the nodes are physically close. The distance metric used for this purpose is derived from the opportunity cost framework [2] which allows one to incorporate and aggregate a multitude of incomparable parameters such as battery life, reliability and security level, bandwidth, etc.

For example, to construct the lowest level of the hierarchy, let us pick the closest node acting as a server for each client. To construct the next levels, we can select next level servers probabilistically by say flipping coins with probability $1/2$, and repeating the process. If a server’s coin comes up heads, then it will act as server also for the next level of the hierarchy, if it is tails then it is finished and will not acquire any more roles. Thus each round of coin flipping involves fewer and fewer of the potential servers. This defines a hierarchy of partitions which are very refined close the leaves and quite coarse close to the tree. Each server node acts as a server for at least level 1, and may act as a server for levels 2-k where k is the highest level the server reached in the coin flipping rounds. (We comment that [8] considers a similar setting where servers get chosen probabilistically on a rotating basis.)

To build the tree, each server selected at each level simply floods the message to determine which nodes will enter its tree. Flooding will establish an uplink pointer at each intermediate node eventually leading to the root of each tree. Each server node will rerun the flooding algorithm periodically for the clusters that they are responsible for. This is required to deal with moving sensors and servers, to adapt to changing resource costs (a node may have its battery drained and so the trees should change), and to detect new sensors. The frequency of these floods is a tradeoff between the overhead of flooding and the quality of routes. Although a fixed frequency is simple, since each server conducts their own flood, the frequency can be adjusted separately and dynamically by each server.

Third, the routing algorithm consists of two operations – routing upstream towards the root of the tree, and routing downstream towards a leaf node. The messages being routed can either be content generated by the leaf sensors, or queries that originate at any node (leaf or internal) in the system.

Upstream routing: In order for a node to reach its parent on the hierarchy, it simply sends the message on its uplink, and each intermediate node relays the message on its own uplink.

Downstream routing: It is more difficult for nodes to relay messages downstream, since intermediate tree nodes cannot keep downlinks for each possible child (this is too expensive for sensors). We consider now the problem of routing from root of a

cluster to any other node in that cluster, and solve this problem recursively. Namely, we assume we can use similar procedure at one level below as a sub-routine.

The data structure needed for the purpose of downstream routing consists of

1. A collection of lateral “bridges” connecting clusters to sibling clusters, as well as
2. A “sibling routing” table indicating which cluster is the next sibling cluster to be traversed on the way to destination sibling cluster.

This data structure is constructed in a preprocessing stage as follows. Bridges are detected upon termination of flooding process as edges whose end-points were “conquered” by different flood-ID’s. Sibling routing tables are built by having each cluster flood a message with its ID thru the whole parent cluster. Upon receipt of such flooded message over a bridge from another cluster, that bridge is designated locally as a preferred gateway en route to the destination cluster, and notification to that effect is sent to the root. The root may receive messages from a number of potential gateway nodes for a particular sibling cluster. In that case it randomly picks one of them. Flooding needs to be performed upon introducing new sibling to the cluster. The total cost of all flooding procedures, in terms of message traffic per edge, is upper-bounded by a term proportional to the product of the number of levels of the recursion, and the degree of tree, i.e., maximal number of sub-cluster in a given cluster.

Finally, we describe how the downstream routing actually works.

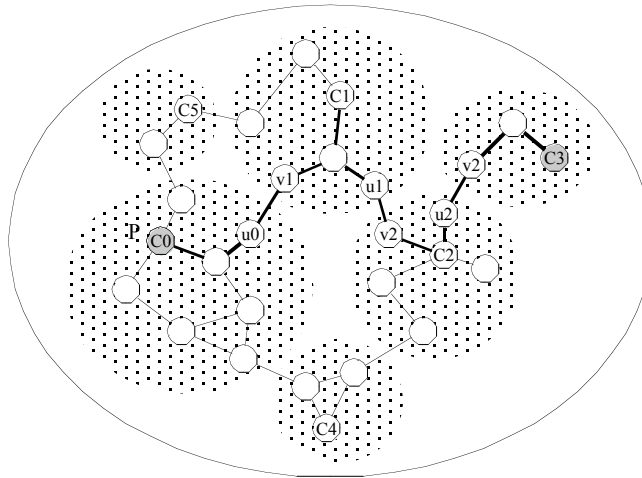


Fig. 1. Downstream and Lateral Routing

In order for a node to reach its child on the hierarchy, it proceeds recursively. Suppose a low-level server C_0 , selected as high-level server P , wishes to reach, say, C_3 who is a child of P . Let C_1, C_2, \dots, C_n also be children of P . The path from C_0 to C_3 starts at the territory of cluster of C_0 and proceeds through the territory of clusters for C_1, C_2 , etc. until reaching cluster of C_3 . Let bridge (u_0-v_1) be the edge emanating from cluster of C_0 into cluster of C_1 on that shortest path. Then, it is enough to send message from C_0 to u_0 inside lower level cluster of C_0 , and instruct it to cross the

bridge (u_0, v_1) . This can be handled recursively since this is taking place on the territory of the lower-level cluster. After crossing the bridge, the message will go to the root of C1 and find out about next bridge (u_1, v_2) into territory of C2, etc. The process will continue till we reach lowest level of the hierarchy (see Figure 1).

With these two routing primitives ready, we can describe the whole process.

1. *Registration of content by climbing the hierarchy*: content generated by a client “climbs the hierarchy” by registering at the parent of that client, and then proceeding to grandparent, etc. This process simply involves forwarding all the information received from children and aggregating information at the servers of given level before continuing to the next level.
2. *Query for contents by climbing the hierarchy*: Query generated at a client climbs the hierarchy exactly like the content registration process. This process simply involves forwarding all the queries received from children. Upon reaching a server on a given level, the query is successful at that level if the content is locally available; otherwise the query is a failure at this level. In the former case the query is stored at a buffer together with the name of the lower-level server who originated it, and is forwarded on an uplink to the next level up, until it eventually succeeds or fails at the top level.
3. *Query for contents by descending the hierarchy*: Once a query succeeds at some level j , it means that that server keeps the desired content, as well as location of lower level child who reported it. The query proceeds downstream recursively, using downstream routing above.

2.3 Extensions to battery-sensitive routing

The above algorithm classifies nodes into just two levels of computing power: servers and clients. In the case of heterogeneous devices with highly different capabilities, it is obvious that higher level servers need to be chosen among stronger devices (ones with higher battery life, higher bandwidth, higher CPU). It is quite possible, however, that servers themselves are not much more powerful than the sensors. In the extreme case, we may consider a completely symmetric peer-to-peer situation where the collectors are the sensors themselves, i.e. clients are no different than servers.

What this means is that clients need to emulate servers. Because of limited battery, and the concentration of traffic at different levels of the hierarchy, the nodes selected to be servers at a given level will quickly run out of battery. This means that their weight will dramatically increase, and the radius of their flooding will be decreased dramatically.

Effectively, this means that the clients they used to be parents of will be reconquered by others floods emanating from higher-battery nodes. This will automatically lead to rotation between different nodes for the role of servers of a given level. If this rotation is accompanied by occasional energy refueling, then the system can keep working indefinitely.

This method applies to a more general setting (with multi-hop connectivity) than the method in [8]. We can prove that our method provides a very precise mathematic guarantees on its performance. As we will see below, this is a serious issue.

2.4 Performance analysis

The efficiency of the above query publishing and processing can be evaluated by considering the cost of retrieving the content through the hierarchy versus the optimum cost of the retrieval.

Consider for example a segment of length n with a hierarchy in the form of a perfect binary tree. Namely, this segment is broken into two segments of length $n/2$, each corresponding to lower level clusters. Consider now a bridge edge from one cluster to another. Note that a query emanating from one endpoint of that edge may traverse the whole tree to find a common ancestor with the other endpoint. Thus, the distance distortion exhibited by such a hierarchy grows linearly with the number of nodes.

It is highly unsettling that by managing to reduce the space overhead through aggregation, from linear to constant, we actually damage our routing efficiency by a linear factor. In a sense, this indicates that we have not dealt with the issue of scalability in a satisfactory manner.

3 Lattices versus trees

This paper suggests a universal mathematical framework for handling the issue of scalability, that we refer to as Hierarchical Redundant Aggregation. The essence of the framework is that it enables one to represent complex structures such as networks with heterogeneous and arbitrarily connected components in a relatively simple format, which is called a “distance preserving lattice”.

In a hierarchical distance preserving lattice, we have levels of hierarchy imposed on the network, where all nodes belong to the lower level of the hierarchy, and just a handful of nodes belong to the top level. Each node has a handful of parents on the next level of the lattice; for simplicity imagine each node has just two parents. One can easily imagine that if each node has two parents, it may have four grand-parents, 16 grand grand-parents, etc. However, a lattice is constructed in such a way that each node has handful ancestors at each level of the lattice hierarchy. Figure 2 shows an example of a zone of sensors of which some have two parents in order to provide relay service for messages from sensors who do not have their server.

Since the tree contains less edges than the cycle, we can say that the tree is an “aggregation” of the cycle. The “distortion” of the aggregate structure is the worst case distance deterioration of the original graph. The above discussion indicate that aggregating a cycle into a tree exhibits distortion whose quality degrades linearly with the size of the network, and thus such aggregation will be considered of “poor quality” for large size networks. One can easily see that any tree will be a poor quality aggregation for very simple graphs, such as, for example a grid graph.

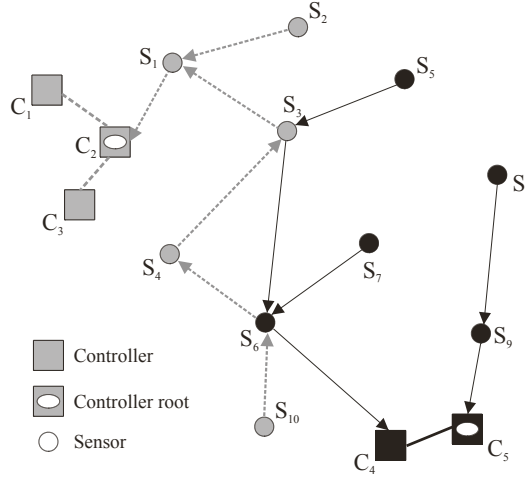


Fig. 2. Sensors with two Controllers and Lattice

The difference between a tree and a lattice is manifested exactly in that in a tree, each node has a single parent in the next layer, and each node has a single ancestor at each level. This difference is crucial in that a lattice is immensely superior to a tree in terms of the ability to capture distances, without the need to introduce exorbitant overhead. In some sense, a lattice can be viewed as a number of trees super-imposed on each other. Observe that two trees can faithfully capture distance on a circle, while it is impossible to achieve this effect with a single tree. Indeed, each tree will fail to contain some edge; and the path on the tree connecting endpoints of that edge is linearly longer than the direct connection. In the next section we provide specific lattice constructions based on the work by Awerbuch and Peleg [1] (we comment that this is the theoretically best data structure for distributed routing).

3.1 Our approach – lattice version

As we have indicated earlier, it is easy to implement a hierarchy where each node enters the next level with small probability, and partition the graph with spanning trees where each node has a single uplink. This is a very scalable data structure, but as we pointed out it cannot possibly be accurate in representing the metric space.

If we are talking about representation of a regular graph, such a heuristic can work. However, even in a regular deployment of sensors, the distance/cost graph will not remain regular for long because due to factors such as battery depletion, any effective embedded sensor system will have to distort a regular graph into an irregular arbitrary structure.

The idea is that adding a couple more uplinks to nodes (which does not really hurt scalability) will have a crucial impact on the distance distortion, reducing the maximal possible distortion from linear to logarithmic. We rigorously support the claim in a form of a concrete algorithm, a theorem, and a mathematical proof of performance guarantees.

The algorithm to construct one level of the hierarchy is very simple. Select a desired range, say r . Our goal is to construct a collection of clusters each one around a single client (who will now be called a server), and a spanning tree for each cluster such that the following properties hold.

Lattice properties

1. Any two clients at distance r from each other will be spanned by a common tree (and thus will be able to locate each other).
2. The tree overlap is at most logarithmic.
3. The radius of each tree is at most logarithmically larger than r .

The lattice algorithm: The algorithm consists of logarithmic number of iterations.

In each iteration, each node selects a random integer, from 1 to $\log n$, with exponentially decreasing probability (next integer half as likely as the current), and floods its ID to radius limited by “distance to live” which is the product of the random value and r . Each node propagates the flood if “distance to live” is positive and the ID of this flood is higher than the highest previously seen ID. In this case, it remembers the node from which it receives this flood as an uplink on the link to the corresponding ID, subtracts its (opportunity) cost from “distance-to-live” and broadcasts the new ID to the neighbors. The set of recorded uplinks forms the desired lattice, which is simply the collection of trees rooted at winning ID’s.

Distributed Lattice Construction Theorem: For every network of size n nodes, an arbitrary set of clients, and every value of radius r , with overwhelming probability, the following properties hold:

1. Lattice properties above are met
2. Each edge is traversed by logarithmic number of messages

Proof: The radius of each tree is at most logarithmically larger than r by construction; thus 3) holds.

Tree overlap is the expected number of maxima in random sequence; the probability of j ’s member to be maximum is $1/j$ and summation of $1/j$ is approximately the logarithm of the number of nodes, proving 2). To prove 1), consider two such nodes x and y at distance r at most. The highest ID that have been seen by x was not seen by y or vice versa. Notice that exponential probability is memoryless, thus with probability 0.5 the highest ID flood that reaches x will also reach y , in which case x and y cannot have different parents, **QED**.

4 Related Work

Some ideas related to imposing hierarchy on top of uniform network of sensors were presented in [8] where the LEACH protocol chooses cluster heads probabilistically on a rotating basis. LEACH assumes that all nodes have the ability to communicate with the final destination, but must pay a higher energy cost to do so. Our cost-benefit method can provide the same advantages as LEACH at the lowest levels of the hierarchy where all the nodes are within maximum range of each other, and can also pro-

vide the more general multi-hop capabilities by using minimum cost forwarding at higher levels of the hierarchy.

Heterogeneous resource routing has also been described in the GEAR project by Estrin et al [GEAR]. We now describe the Geographical and Energy Aware Routing (GEAR) algorithm. GEAR uses a geographical and energy aware neighbor selection heuristic to route the packet towards the target region. At least from the algorithmic point of view, we believe that we should be able to greatly improve on the performance of such a solution by using a cost-benefit framework which will unify bandwidth-based and energy-based optimization in a provable manner.

Also utilizing geographic information, but for a different purpose, GAF described in [17] is built using a different methodology of evaluating power usage. The paper offers evidence that the power consumed while idle/listening is significant in comparison to that consumed while transmitting or receiving, and in the low traffic case dominates the power consumption of the network. This results in an accounting methodology that does not ignore idle/listening power consumption and a strategy of utilizing the sleep capabilities of wireless transceivers in order to save some of the power used while nodes are idle. GAF uses location information and a virtual grid in order to approximate a maximal independent set of nodes that must stay awake. Because this approach relies on finding sets of nodes that are all interchangeable, it only provides savings for the most trivial cases, and isn't robust in the case of complex radio propagation phenomena. The opportunity-cost framework provides an alternative strategy where the cost of keeping a node awake can be compared with the benefit it provides to the network. This analysis can be used to select a randomized duty cycle appropriate for each node.

5 Conclusion

We have presented a general-purpose highly scalable distributed algorithm for content and query routing on heterogeneous sensor networks. This algorithm has provably strong bounds on the distance distortion it produces, and provides logarithmic message costs for both content distribution and query routing. We believe the lattice routing architecture offers substantial potential for providing a general-purpose routing architecture that can incorporate much of the research on sensor network routing (by choosing different cost metrics, the lattices formed will adapt to many different researchers specific algorithms).

References

1. B. Awerbuch and D. Peleg. Routing with polynomial communication-space trade-off. *SIAM Journal on Discrete Mathematics*, 5(2):151-162, May 1992.
2. B. Awerbuch and Y. Azar and S. Plotkin: Throughput-Competitive On-Line Routing. Proceedings IEEE FOCS 1993, 32-40
3. D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks.

4. N. Bulusu, D. Estrin, L. Girod, and J. Heidemann. Scalable coordination for wireless sensor networks: Self-configuring localization systems. In *The 6th International Symposium on Communication Theory and Applications*, July 2001.
5. Demand Source routing by Johnson and Maltz , 1996.
6. D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *The 5th Annual International Conference on Mobile Computing and Networks*, August 1999.
7. D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly resilient, energy efficient multipath routing in wireless sensor networks. *Mobile Computing and Communications Review*, 1(2), 2002.
8. W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocols for Wireless Microsensor Networks. Proc. Hawaiian Int'l Conf. on Systems Science, January 2000
9. W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *The 5th ACM/IEEE Mobicom Conference*, August 1999.
10. J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Symposium on Operating Systems Principles*, October 2001.
11. C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks.
12. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *The 6th Annual International Conference on Mobile Computing and Networks*, August 2000.
13. D. B. Johnson, D. A. Maltz, and J. Broch. *DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks*. in *Ad Hoc Networking*, chapter~5, pages 139--172. Addison-Wesley, 2001.
14. S. Kumar, C. Alaettinoglu, and D. Estrin. Scalable object-tracking through unattended techniques. In *The 8th International Conference on Network Protocols*, November 2000.
15. B. Krishnamachari, D. Estrin, and S. Wicker. Modeling data-centric routing in wireless sensor networks.
16. Y. Xu, J. Heidemann, and D. Estrin. Adaptive energy-conserving routing for multihop ad hoc networks. Technical Report Research report 527, USC/Information Sciences Institute, October 2000.
17. Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad-hoc routing. In *The 7th ACM/IEEE International Conference on Mobile Computing and Networking*, July 2001.
18. Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, UCLA Computer Science Department, May 2001.
19. W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *The 21st International Annual Joint Conference of the IEEE Computer and Communications Societies*, June 2002.