A Robust and Light-Weight Routing Mechanism for Wireless Sensor Networks

Jing Deng, Richard Han and Shivakant Mishra
Department of Computer Science, University of Colorado
Campus Box 0430, Boulder, CO 80309-0430, USA.
{jing|rhan|mishras}@cs.colorado.edu

Abstract

This paper presents a light-weight, dependable routing mechanism for communication between sensor nodes and a base station in a wireless sensor network. This mechanism tolerates failures of random individual nodes in the network or a small part of the network. The mechanism is light weight in the sense that every node does only local routing maintenance, needs to record only its neighbor nodes' information, and incurs no extra routing overhead during failure-free periods. It dynamically discovers new routes when an intermediate node or a small part of the network in the path from a sensor node to a base station fails.

1. Introduction

Wireless sensor networking that combines data sensing, computing, and communication functions has been gaining tremendous popularity in recent days. Several real-world applications have already been designed, implemented and deployed [1][9]. A wireless sensor network (WSN) consists of a large number of sensor nodes and one or more base stations. A base station acts as a gateway to connect a WSN to the outside world, e.g. an end user or the Internet. Individual sensor nodes sense their environment, and transmit the sensed or processed data to a base station via a multi-hop network consisting of several sensor nodes. The base station in turn transfers the data to the WSN users.

Although each individual sensor node is highly constrained in its computing and communication capabilities, a complete WSN is capable of performing complex tasks. An attractive feature of a WSN is the ease of deploying redundancy. Because of the presence of a large number of relatively inexpensive sensor nodes, important dependability mechanisms such as node and route replication are viable in a WSN. So, while individual sensor nodes are highly vulnerable to failures by battery drain, outside damages, or security attacks, a complete WSN can be built to tolerate/mask these failures. In this paper, we propose a light-weight, fault-tolerance routing mechanism for wireless sensor networks.

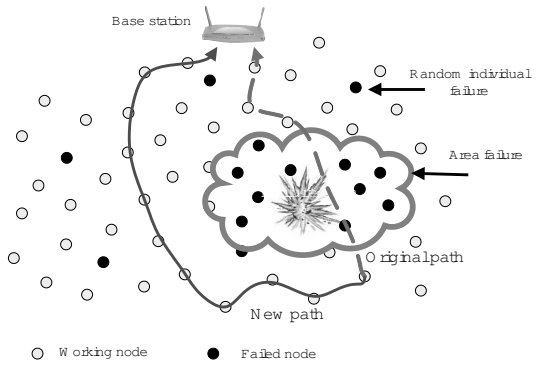

Figure 1. Node failure in WSN

Roughly speaking, a WSN can suffer from two types of failures: *random node failure* and *area failure* [6][11]. Every node in a WSN has similar probability to suffer from a random node failure, which is typically caused by battery drain or some internal problem in the node. On the other hand, an area failure results in a failure of all nodes within a certain geographical area. This is mostly caused by outside accidents, such as a bomb explosion, fire, successful denial-of-service attacks, and so on. Figure 1 illustrates these two types of failures in a WSN.

Several routing protocols for a WSN have been proposed [4, 10, 8, 3, 14, 15, 12, 13]. Three different methods have been used to maintain routing paths in the presence of node failures. In the first method, routing paths are reconstructed periodically. For example, in a simple beacon protocol [7], a base station periodically broadcasts a beacon message. By receiving a beacon message, a node receives an up-to-date routing path to the base station. Reconstruction of routing paths is expensive in this method and consumes lots of energy. In addition, since re-construction is not on-demand, a nodes has to wait until the beacon to update the routing information on a node failure.

In the second method, multiple routing paths are used to transfer data. The key idea is that unless every path from a sensor node to a base station is broken by a failed node, data can be transmitted to base station. The multipath version of directed diffusion [6] uses this strategy. In ARRIVE protocol, copies of the same data is sent through different paths with certain probability to avoid data loss caused by single node failure or failure area [11]. In INSENS [5], multiple disjoint paths are built to bypass a failed node. This method can result in increased energy consumption and packet collisions, because data is sent along multiple paths, irrespective of whether there is a node failure or not. Also, this method cannot guarantee bypassing an area failure.

In the third method a routing path is selected probabilistically. In this method, a node chooses another node to forward a packet with certain probability. Since there is no fixed path to forward data, a failed node can't block all packets from a sensor node to a base station. The ARRIVE routing protocol [11] uses this strategy to forward multiple copies of the same data. In rumor routing protocol [2], instead of flooding query to whole network, a query is sent through a random path until it meets event area. This probabilistic approach provides some fault tolerance, but data loss is still possible with some probability. In addition, the random chosen path is not energy efficient.

Recently, Woo et al. investigated the challenges of multihop routing in wireless sensor networks and proposed a routing scheme based on node's neighborhood link estimates [13]. This protocol is for a many-to-one, data collection routing scenario in WSN. A sensor network can quickly react to node failures and data transmission range changes, and find new routing path for sensor nodes. To do this, a sensor node needs to periodically broadcast its routing information, or periodically probe its neighbor nodes' routing information. In addition, it needs to maintain a table which contains its neighbor nodes' routing information.

In this paper, we propose a robust and light-weight routing mechanism that can be incorporated in any routing protocol for WSN to make it fault tolerant. It dynamically repairs a routing path between a sensor node and a base station. In contrast to [13], a node stores only its parent node routing information, and asks for neighbor nodes routing information when parent node is inaccessible. When an original routing path is broken, a node selects a new path from its neighbor nodes. This light-weight routing mechanism tolerates both random node and area failures.

2. Protocol Description

2.1. Assumptions

In this paper, we focus on how each sensor node maintains its routing path to a base stations. We assume that the initial routing mechanism from each sensor node to a base station has already been set up. This can be done using a number of protocols that have been proposed in the past, e.g. the TinyOS beacon protocol discussed below. In particular, we assume that each node already has a path to the base station, and knows its parent node, neighbor nodes and the number of hops it is from the base station. This information can be initialized by using the TinyOS beacon protocol for setting up routing paths. In this protocol, the base station floods a beacon message in the network. When a node first hears the beacon message, it records the sender of that beacon message as its parent node and forwards the beacon message to all of its neighbor nodes. When a node needs to send/forward a message to the base station, it sends the message to its parent node. The parent node in turn forwards the message to its parent node, and so on, until the message gets to base station. A key problem with this protocol is that it is not fault tolerant. If the parent node of a sensor node fails, the sensor node cannot communicate with the base station.

2.2. Path Repair Algorithm

The basic idea to repair routing paths in case of random node or area failures is quite simple: every node monitors its parent node. When it finds that parent node has failed, it asks its neighbor nodes for their connection information. It then chooses a new parent node from its neighbor nodes based on this connection information. As shown in Figure 1, the mechanisms can tolerate node failures and routes a message circumventing a the failed nodes.

This mechanism consists of four parts: the *failure detection*, *failure information propagation*, *new parent detection*, and *new parent selection*. First, a node detects if its parent node is alive and if the parent node can connect to base station. This part is called *failure detection*. If a node s detects that its parent node works well, it won't do any maintenance work. If there are some problems in parent node, such as node failure or disconnected to base station (probably one of parent node's ancestor node is failed), node s informs its children nodes about the failure, which is called *failure information propagation*. In addition, s requests the connection information from its neighbor nodes since it needs to choose a new parent node from them. This part is called *new parent detection*. After collecting information from its neighbor nodes, s decides a new parent node based on the information it collected. This part is called *new parent selection*.

We denote a as the node who tries to maintain its route path. Node p is a's parent node.

1 node a sends $PROBE$ message to its parent node p, and set a timeout (*timeout_prob*) for $BACK$ message from p.

$$PROBE : a \rightarrow p : probe\_prt$$

2 if the p can hear the $PROBE$ message, it will reply a $BACK$ message. The $BACK$ message contains the information that whether p connects to base station or not, and if it is connected, the hops to base station. If p connects to base station, it sends $BACK\_Y$ message back to a. The format of $BACK\_Y$ is

$$BACK\_Y : p \rightarrow a : connected||hops$$

If p cannot connect to base station, it sends $BACK\_N$ back to a:

$$BACK\_N : p \rightarrow a : broken||broken\_hops$$

the $broken\_hops$ represents the distance to untouched ancestor node. If p cannot connect to its parent node $p.parent$, the $p.broken\_hops$ is set to 1. Otherwise,

$$p.broken\_hops \leftarrow p.parent.broken\_hops + 1$$

3a if a receives $BACK\_Y$ from p, a resets its hops as parent p's hops plus one: $a_{hops} \leftarrow hops + 1$. If a node's hops beyond a maximum threshold value, it sets itself unconnected: $a_{hops} \leftarrow \infty$.

3b if p is dead or its signal is jammed, it cannot reply $BACK$ message within $timeout\_prob$. if a cannot receive $BACK$ message from p within $timeout\_prob$, a knows that it cannot connect to base station through p. Then it broadcasts a $RQST$ message to all of its neighbor nodes to find a new parent node.

$$RQST : a \rightarrow NEIGHBORS : request\_parent$$

3c If a receives $BACK\_N$ from p, a knows that p cannot connect to base station at that moment. Instead of broadcasting $RQST$ message immediately, a waits a timeout $timeout\_prt$ before sending $RQST$. The $timeout\_prt$ depends on the value of $broken\_hops$ from $BACK\_N$ message. This strategy gives parent node p some time to find its new parent node. a will set its $broken\_hop$, and propagate it when its children nodes send $PROBE$ message to a.

4 when one of a's neighbor node c receives $RQST$ message from a, and if c can connect to base station, it sends a $RPLY$ message back to a. $RPLY$ message contains the ID of c's parent node, and c's hops to base station:

$$RPLY : c \rightarrow a : connected||c\_hops||c.parent$$

if c cannot connect to base station, it won't send any message back to a. Instead, it records a as one of its $RQST$ senders. (here, a's children nodes won't sends $RPLY$ message back to a since it is not necessary.) If a hasn't got any $RPLY$ message from its neighbor nodes, it will re-send $RQST$ after a certain timeout $timeout\_rqst$.

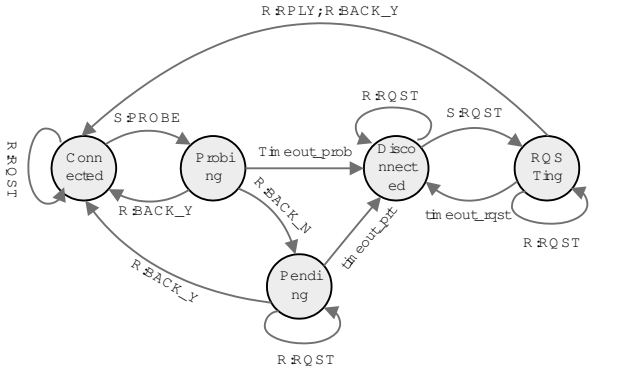

Figure 2. Protocol finite state machine.

5 When a receives $RPLY$ messages from its neighbor nodes, if the $RPLY$ message says that the sender connects to base station, a records the sender as a parent candidate. Finally, a selects its new parent node whose hops to base station is smallest among all candidates. After it selects parent node, a sets its hops as its parent node's hops plus one: $a_{hops} \leftarrow a.parent_{hops} + 1$. If a ever received $RQST$ message from its neighbor nodes, it will send $RPLY$ back to the $RQST$ senders.

In figure 2, we present a formal description of this mechanism with a finite state machine (FSM). This FSM shows the major state translation except the caching/precessing of RQST requests. We use $x : y$ to describe the state translation condition. x denotes the action of events: R means receives a message, S means sends a message. y denotes the content of message.

3. PROPERTIES

3.1. Random node failure and area failure

The proposed mechanism is robust in finding new paths under random node failure and area failure. Figure 3 demos how a node find alternative path when its parent node is failed. In figure 3, p is a failed node, showed as a black node. When p's child node a detects that it cannot connect to p by running step 1, a broadcasts $RQST$ message to its neighbor nodes (as described in step 3b). If any of a's non-child neighbor nodes can connect their parent nodes, they will send $RPLY$ message back to a (in step 4). This figure demos that a chooses c as its new parent node from the $RPLY$ messages, and then a has a new path to base station.

Figure 4 shows that the node are randomly failed in the network. The black nodes are failed nodes. The dash-anchor lines point connect original child node to original parent node, and the solid-anchor lines show the $RPLY$

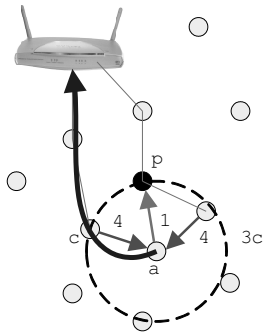

Figure 3. Demo of bypassing a failure node

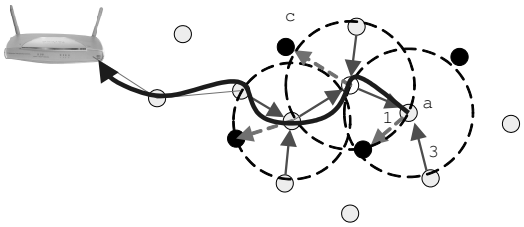

Figure 4. Bypassing random failed nodes

messages corresponded to $RQST$ messages. With our proposed mechanism, Node a finds a new path to base station and bypasses several failed nodes on the path.

Figure 5 demos the case that the nodes within a certain area are all failed. This may be caused by some accidents, i.e, fire, a bomb, or a signal jamming attack. This type of failure is called area failure. When it happens, the nodes just close to the failure area will send $RQST$ messages to their neighbor nodes. In the beginning, some nodes choose other nodes along the failure edge as their parent nodes. That is because these nodes may detect the failure area at slight different time. But quickly, the nodes just behind the failure area will detect that their neighbor nodes are also disconnected to base station. We call this area as "block area". Because of routing update inconsistency, some nodes may form routing loop in the "block area", which we will discuss in next section. In the edge of the failure area, which we call "edge area", nodes will find the real path to base station, and the routing information of these nodes will eventually affect the nodes in "block area" and connect them to base station.

3.2. Routing Loop Discussion

3.2.1. Loops In our proposed mechanism, every node decides its path based only on local information, such as its parent node and neighbor nodes' routing information. So, it

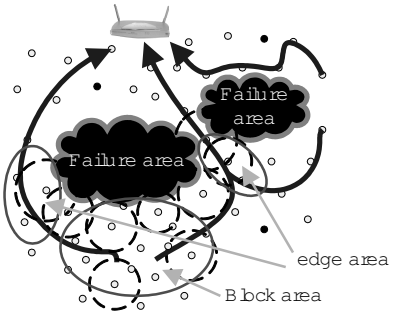

Figure 5. Bypassing area failure

is possible to form a loop in the routing path. Because the $RPLY$ message contains the parent node of $RPLY$ sender, a node can find and avoid the short loop which only contains 2 or 3 nodes. However, longer loops cannot be prevented. In general, an occurrence of a loop is more likely in case area failure than random node failures.

When an area failure occurs, some nodes detect their parent failures and send $RQST$ messages, and some nodes that haven't yet detected the failure keep their old routing information. This routing information inconsistency can create loops. However, if there is a loop, the hop count of that path will continuously increase, and eventually it will be higher than the hop count of paths of nodes that have real paths to the base station. If a node finds its hops continually (and regularly) growing, it will begin to detect all its neighbor nodes' hops. These nodes that have real paths will attract the nodes on the loop with their low $hops$ value and finally break the loop.

We believe that the loop problem in sensor network routing is not as serious as that in the Internet routing or traditional mobile as hoc routing. This is because we are only concerned with communication between sensor nodes and a base station in a WSN, as opposed to communication between any two nodes in other networks. If the base station is not isolated, and if there is at least one node in the loop whose neighbor node connects to base station, the indefinite increase in hop count will cause this node to use its neighbor node as the parent node at some point in time and hence break the loop. Another problem caused by loops is energy consumption and increased packet delay/loss. Nodes in a loop may waste their power by continually forwarding packets.

3.2.2. Loop Elimination We don't use sender ID and originating sequence number to detect loop since that requires a node s to memorize lots of history information if there are lots of nodes sending packets to base station through s. In stead, we propose the following mechanism to eliminate loop. Suppose there is a loop $a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_k \rightarrow a_1$. This loop exists because there is a node (a_k) that

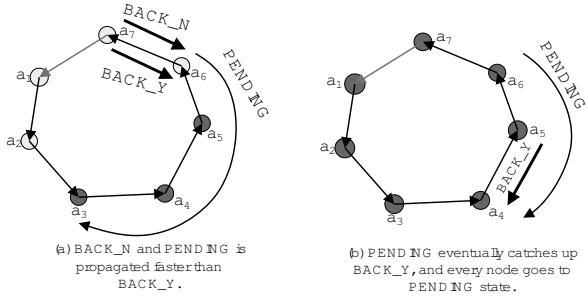

Figure 6. Elimination of loop

finds that its original path is broken and it can connect to a_1. Before choosing a_1 as its new parent node, a_k needs to broadcast $RQST$ to its neighbor nodes, and so a_{k-1} will know a_k's path is broken. If a_{k-1} and its downstream nodes continually inform their downstream nodes the path broken event, the "path broken" information will quickly propagate to all downstream nodes. At the same time, a_k accepts a_1 as its new parent node and sends new $hops$ information to its downstream nodes. Although the "new hops" information will eventually propagate to all downstream nodes, its propagation speed is much slower than "path broken" event, since a child node gets "new hops" information after it sends $PROBE$ message and receives $BACK\_Y$. If there is a loop, from a_1 through a_k to a_1, the "path broken" event will get to a_1 and continue to reach a_k, and eventually it will catch "new hops" information. At that time, every node on the loop will get "path broken" event and the path of the loop will disappear.

One way to implement above strategy is that in step 3c, after receiving $BACK\_N$, node a immediately sends $PENDING$ message to its children nodes. The format of this message is:

$$PENDING : a \rightarrow CHN : pending||pending\_hops$$

Initially, a sets $pending\_hops$ as 1. When a node receives $PENDING$ from its parent node, it increases $pending\_hops$ by 1 and forwards this message to its children nodes immediately. This way, the "path broken" information will spread to all downstream node very quickly. However, although the $PENDING$ propagation prevents loop, it may also generate the $PENDING$ message storm to downstream nodes. To prevent $PENDING$ storm, a node can slow down $PENDING$ message forwarding. It can wait a short time out ($timeout\_pend$) before forwarding a $PENDING$ message, $timeout\_pend < timeout\_prob$. In addition to avoiding loop, a $PENDING$ message can also be used to control packet sending rate, i.e. downstream nodes will slow down or stop sending packets after they know that the path is temporary broken.

4. Variations and Extensions

4.1. Generalization of Our Mechanism

In section 2, we described a basic robust and light-weight routing algorithm for WSN, which is composed of four parts: *failure detection*, *failure information propagation*, *new parent detection*, and *new parent selection*. In *failure detection*, a node detects if its parent node has failed. We use $PROBE$ and $BACK$ messages to detect failure. In *failure information propagation*, a node tells other nodes about the failure information. $BACK\_N$ and $RQST$ messages are used for this purpose. In *new parent detection*, a node finds out information about new parent candidates. $RQST$ and $RPLY$ messages are used to find new parent candidates. Finally, in *new parent selection*, a node uses appropriate metrics to choose a new parent node from candidate nodes. In the basic mechanism, a node uses number of hops to base station as the metric.

In real applications, all four parts can be modified to adapt to the requirements of different scenarios. For example, in Section 3, we changed the way failure information is propagated to eliminate routing loop. For failure detection, we can attach a piggyback a $PROBE$ message in normal data packets, and $BACK$ message can be used for reliable hop-by-hop data transmission.

4.2. Using Different Metrics

An important part of our mechanism is the metrics used for new parent node selection. All nodes must use a common metrics to evaluate their routing cost to the base station. This metrics must be such that its value decreases monotonically as you get closer to the base station. Every node can simply use a greedy algorithm to select its parent node based on this metrics value. Number of hops is one type of metric that satisfies this property. Any other metric that satisfies this property can also be used.

4.2.1. Metrics Based on Location Information If a node can get location of its neighbor nodes (by using GPS, directional antenna or other techniques), then it can choose a parent node node based on the location of the failed nodes. For example, when it finds that most of its neighbor nodes in one direction have failed, it concludes that there is an area failure in that direction. In that case, it will choose a new parent node based not only on hops cost, but also on its location relative to the failed area.

4.3. Directed Diffusion

The proposed robust and light-weight routing mechanism can be used in directed diffusion based routing algorithms. In directed diffusion routing algorithm [10], a sink

node disseminates its $interest$ to the network. When corresponding source node gets the $intrest$, it sends $events$ data back to sink along the path through which $interest$ disseminated. Then the sink reinforce a path that connects sink and source nodes. This reinforcement is based on the cached $events$ propagation information. Every new node on the path does reinforcement until the path gets to source node. If a link is broken, a node can find alternate path by running reinforcement again. Since the dissemination of $intest$ passed a large area of nodes between sink and source, the nodes within the area can get and keep the cost metrics to sink. When the reinforced path is broken, other nodes on the path can run this scheme to find another path towards sink.

4.4. Node Join and Network Re-Construction

In this paper, we focus on node failure problem in WSN. However, we can extend our scheme to deal with new node joins and recovery of failed node. When a new node is added in the network, it broadcasts a message to find its neighbor nodes and their hops to base station. Then this node can choose its parent node and join the network. In addition, the new node may change other nodes' paths to the base station. Some nodes may have shorter path to the base station through new node. Here, we use conservative strategy for a node to change its parent from a longer path to a shorter path because that is useful to prevent loop, and prevent malicious node from sending forged hops information. When a node finds that its neighbor nodes has a shorter hops to base station, it sends two copies of a message to base station, one through that node and the other through its current parent node. If it receives the feedback message from its neighbor node earlier that from its parent node, then it consider to change its parent node.

If there are lots of node failure and new node joining in the network, our scheme can still build routing paths for alive nodes but the paths may not be efficient. In this situation, it is better to use base station to send beacon message to reconstruct the routing paths in the network.

5. Conclusion and Future Work

In this paper, we have presented a light-weight, dependable routing mechanism for communication between sensor nodes and a base station in a wireless sensor network. This scheme tolerates failure of random individual nodes in the network or a small part of the network by dynamically discovering new routes when nodes fail. The proposed mechanism is generic in the sense that it can be incorporated in several routing protocols to make them fault tolerant. In the future, we plan to experiment with this mechanism, including a simulation and implementation, to evaluate its performance and usability in a real sensor network application.

References

[1] H. Abrach, S. Bhatti, J. Carlson, H. Dui, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han. Mantis: System support for multimodal networks of in-situ sensors. WSNA'03, San Diego, CA, USA, September 2003.

[2] D. Braginsky and D. Estrin. Rumour routing algorithm for sensor networks. WSNA'02, Atlanta, GA, USA, September 2002.

[3] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *ACM Wireless Networks Journal*, 8(5):481–494, 2002.

[4] D. E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo. A network-centric approach to embedded software for tiny devices. In *1st international workshop on embeded software*, Tahoe City, California, USA, October 2001.

[5] J. Deng, R. Han, and S. Mishra. The performance evaluation of intrusion-tolerant routing in wireless sensor networks. IPSN'03, Palo Alto, CA, USA, April 2003.

[6] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly resilient, energy efficient multipath routing in wireless sensor networks. MC2R, 1(2), 2002.

[7] D. Ganesan, B. Krishnamurthy, A. Woo, D. Culler, D. Estrin, and S. Wicker. An empirical study of epidemic algorithms in large scale multihop wireless networks. Technical Report IntelIRP-TR-02-003, Intel Research, March 2002.

[8] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. HICSS'00, Maui, Hawaii, USA, January 2000.

[9] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Cullar, and K. Pister. System architecture directions for network sensors. ASPLOS'00, Cambridge, MA, USA, November 2000.

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robuts communication paradigm for sensor networks. In *the Sixth Annual International Conference on Mobile Computing and Networking*, Boston, MA, USA, August 2000.

[11] C. Karlof, Y. Li, and J. Polastre. Arrive: Algorithm for robust routing in volatile environments. Technical Report UCBCSD-02-1233, Computer Science Department, University of California at Berkeley, May 2002.

[12] B. Karp and H.T.Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. MobiCom'00, pages 243–254, Boston, MA, USA, August 2000.

[13] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges fo reliable multihop routing in sensor networks. SenSys'03, Log Angeles, CA, USA, November 2003.

[14] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. MobiCom'03, pages 70–84, Rome, Italy, July 2001.

[15] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks. Technical Report UCLACSD-TR-01-0023, Computer Science Department, University of California at Los Angeles, May 2001.