Energy Efficient Data Management for Wireless Sensor Networks with Data Sink Failure

Hyunyoung Lee*, Andreas Klappenecker†, Kyoungsook Lee*, Lan Lin*

*Department of Computer Science, University of Denver, Denver, CO 80208, U.S.A.
{hlee,klee,llin}@cs.du.edu
†Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, U.S.A.
klappi @cs.tamu.edu

***Abstract*— This paper proposes an energy efficient protocol for sensor data management. The protocol employs replicated data sinks to achieve (1) resiliency to data sink failure, and (2) efficiency in storing and retrieving sensor data. A simple address assignment scheme is introduced that partitions the sensor field into cells, where each cell contains one data sink and all sensors that are closest to this data sink. It is shown that this scheme is scalable and resilient against data sink and sensor node failures. Furthermore, the scheme has a reasonably low message complexity and a high energy efficiency.**

I. Introduction

There is an upsurge of interest in wireless sensor networks, since they have many important applications in everyday life, ranging from monitoring and detection to space exploration. In this paper, we propose an energy efficient protocol for storing and retrieving sensor data. Our protocol provides fault-tolerance in the presence of data sink and sensor failures. As a result, our protocol can maximize the overall life of the sensor network.

Sensors are usually very simple units that are equipped with a sensing functionality. As indicated by Moore's law [16], one can expect that wireless sensors become smaller, cheaper, and more powerful. Sensors can even carry out simple computations and communicate with each other.

However, a wireless sensor node has limited resources since it typically runs on battery power and usually has a very small memory space. Thus, sensing devices must operate under severe resource contraints and one of the foremost goals is to minimize the energy consumption. Therefore, there is a need for an energy-efficient communication scheme to store and retrieve a vast amount of sensor data. Furthermore, in many applications, the sensing devices are placed outdoors, resulting in a vulnerability to various noises and errors. With these characteristics of a wireless sensor network, we consider the following naturally arising questions:

1) What kind of data storage and retrieval structure in a wireless sensor network is energy-efficient?
2) How can we make the wireless sensing system fault-tolerant, when sensor nodes and data sinks may fail?
3) How can we achieve scalability in wireless sensor data management so that the sensor system can be easily expanded by deploying new sensors and even adding new data sinks?

As an effort to answer these questions, we designed a protocol based on ideas inspired by *de Bruijn digraphs* [1] and *Voronoi diagrams* [2]. It is well-known that de Bruijn networks can provide efficient routing among large number of nodes. Our routing scheme imitates certain aspects de Bruijn routing, but is simpler, more flexible, and dynamically reconfigurable. In our scheme, the address of a sensor node already indicates the length of the path to the closest data sink.

Recall that a countable set P of points in the Euclidean plane $\mathbf{R}^2$ leads to a partition of the plane in terms of Voronoi cells, where each cell contains exactly one point p of P and all points in $\mathbf{R}^2$ that are closer to p than to any other point in P. Inspired by this geometric notion, we partition the sensor network into different cells, where a cell contains one data sink s and any sensor that has a smaller hop count to s than to any other data sink. If a sensor has the same hop-count to two or more data sinks, then we agree that this border node will belong to the cell of each of these data sinks. Figure 2 (c) illustrates this concept.

By partitioning the sensor network into such cells, we obtain scalability and improve the energy efficiency. If some data sink or sensor node fails, then our protocol dynamically re-assigns the cells to provide resilience against such failures.

Our communication architecture uses a hybrid model that effectively utilizes a variation of the peer-to-peer communication paradigm among the sensors, and a variation of the client-server paradigm between the sensors and the data sinks. The wireless sensors act as clients in the networked sensor system and the data sinks act as servers. The data sinks process the collected data and return feedback control data to the sensor nodes.

The remainder of this paper is structured as follows. In Section II, we discuss related work and give some background on de Bruijn digraph routing. In Sections III and IV, we specify the system model and describe our protocol. In Sections V and VI, we analyze the properties of our protocol and conclude the paper.

II. Related Work

Our work is related to two intertwined themes in wireless sensor networks: routing and data aggregation. Numerous architectures and protocols have been proposed to solve both problems at the same time.

Our scheme requires an initial flooding of messages in the sensor field to establish the routing paths. This step is somewhat similar to directed diffusion [7], a mechanism that uses limited flooding of queries towards events and sets up reverse gradients for the best path. One fundamental difference is that directed diffusion is designed for the single data sink scenario, whereas our protocol can serve multiple data sinks.

GPSR is an efficient routing scheme that relies on the localized nodes and restricts flooding to a geographical region [9]. One drawback of this approach, however, is its assumption that the locations of the sensor nodes are known to all nodes in the network. We designed our protocol such that knowledge of locations is not required.

SHORT is a self-healing, path- and energy-aware routing framework that shows a good performance with the reduced energy costs [4]. In a path-aware scheme, shorter paths are found by connecting non-adjacent nodes on a path that are within communication range of each other. In an energy-aware scheme, a routing path is switched when the energy of the nodes on the path is running low. By letting the neighboring nodes of a route, together with the on-route nodes, monitor the route, up-to-date information of local topology and link quality can be exploited. Our work resembles their approach regarding self-healing and energy-efficiency. In our case, the routing of messages to a data sink is optimal, and we take advantage of shortcuts in peer message routing, though without introducing much overhead.

Demirbas, Arora, and Mittal [3] presented a clustering service, called FLOC, that can achieve efficient and scalable control in large-scale ad hoc wireless sensor networks.

To achieve high energy efficiency and resiliency, role-based hierarchical self-organized networks are explored in [10]. Depending on their connectivity and sensing capability, sensor nodes are assigned the role of data collection and data dissemination. Based on certain metrics, the network is partitioned into sensing zones, in which the sensor nodes collaborate to achieve a sensing objective. Like our scheme, this approach relies only on local information. However, as a hierarchy-based architecture, this approach is vulnerable to failures, especially when particular roles are prone to become points of failure. The authors mention that systematic rotation of roles among the nodes can resolve this problem. A periodically repeated role assignment scheme is proposed in [5], for Bluetooth-based sensor networks.

ACQUIRE [12] is an active query forwarding mechanism in a sensor network. A query packet is forwarded through the network that follows a random or guided path. At each step, a node, upon receiving a query, performs an update to gather data from all of its neighbors within a look-ahead of d steps. As this query progresses through the network, it is gradually resolved into smaller components until it is completely solved and is returned back to the querying node. This approach works at its best for one-shot, non-aggregate, complex queries for replicated data.

TAG is a high-level abstraction of a declarative interface for data collection and aggregation in wireless sensor networks of TinyOS motes [11]. It realizes a distributed query aggregation scheme that is sensitive to resource constraints and can cope with lossy communication of wireless sensor networks.

Finally, we should mention mobile agent based systems, such as [15], where agents exchange data with nearby sensors or access points that they encounter as they pass by. The advantage of such an approach is that less infrastructure is required than in other methods and that there is no overhead caused by packet routing. When the density of mobile agents is sufficiently high, the system is more robust than a fixed network. The primary drawback of such a system is that the latency is high, so it is not suitable for all applications. Unexpected failures such as loss of mobile agents or limitations on mobility can compromise the fault-tolerance of such a system.

A. De Bruijn Digraphs

We recall the basics of de Bruijn digraphs, see [1] for details. Routing in such graphs is a well-studied problem, see, for instance, the references [6], [8], [13], [14], [17].

Let h and k be integers ≥ 2. The *de Bruijn digraph* $B(h,k)$ has vertex set $V = \{0,1,\ldots,h-1\}^k$, and there is an edge from vertex $\underline{a} = (a_1,\ldots,a_k)$ to vertex $\underline{b} = (b_1,\ldots,b_k)$ if and only if $a_i = b_{i+1}$ for all i in the range $1 \leq i \leq k-1$. Thus every vertex has an out-degree of h, and the diameter of $B(h,k)$ is equal to k. Figure 1 illustrates the digraph $B(2,2)$.

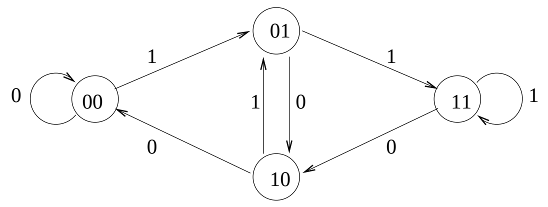


Fig. 1. The de Bruijn digraph B(2,2).

One possible routing scheme in a de Bruijn digraph works as follows. Suppose that the destination address is $\underline{b} = (b_1,\ldots,b_k)$ and the source address is $\underline{a} = (a_1,\ldots,a_\ell,b_1,\ldots,b_{k-\ell})$, where $(b_1,\ldots,b_{k-\ell})$ is the longest prefix of $\underline{b}$ at the tail of $\underline{a}$. Then the routing can be done by left-shifting the source address ℓ times, inserting one digit of the destination address in each step, starting from digit $b_{k-\ell+1}$.

We note that it is not possible to use de Bruijn routing in a sensor network, since the de Bruijn digraph cannot, in general, be embedded into the available communication topology of the sensor network. However, we can retain much of the routing principle for the communication of sensor nodes to data sinks, as we will see in Section IV.

III. System Model and Assumptions

Let $W(t,n)$ denote a wireless sensor network with t replicated data sinks $D = \{d_1,\ldots,d_t\}$, and n sensors $S = \{s_1,\ldots,s_n\}$. The data sinks are sensor-oblivious, which means that a sensor can store and retrieve data to and from any data sink. We assume that the t data sinks are reasonably regularly deployed over the sensor field.

We make the following assumptions about the cost for an interaction between a data sink and a sensor.

- The cost (energy consumption) of storing and retrieving data is the same at every data sink.
- The cost of sending and receiving data to and from a data sink can be computed by the hop count in the routing path to the data sink times a fixed cost per hop.

Each sensor tries to minimize the cost of storing and retrieving data by communicating with the nearest data sink, where the distance from a sensor to a data sink is measured in terms of hop counts. It follows that the sensor network is partitioned into cells such that the sensors in the same cell communicate with the same data sink. We call the nodes on the border of two or more cells "border nodes".

We assume that unique identifiers (ID) are given to data sinks. We also assume that every sensor node has a unique identifier, such as a MAC address. There is no functional difference among data sinks, that is, they all act as final data storage and gateway to the outside networks. Data can be sent to any of the data sinks as long as the data sink is alive. We assume that the wireless sensor nodes as well as the data sinks are stationary, i.e. not mobile. We also assume that the data sink servers know the total number n of sensor nodes, and that only a subset of the sensors are within one-hop range from the data sinks (if all the sensors are within a radio range from the data sinks, then there is no need for routing).

The wireless signal (message) that a sensor node sends is broadcast within the radio range, that is, every node within the radio range of a sensor node i will hear the messages broadcast by i. Delivering a message requires more processing power than receiving a message. Therefore, in the design of our protocol, we try to minimize the redundant delivery of messages without compromizing the fault-tolerance in data transmission.

IV. The Protocol

In this section, we describe our protocol for energy-efficient, fault-tolerant data storage and retrieval, without relying on any geographic or physical location information of the sensors as well as the servers. Our protocol uses the five types of messages:

1) The initialization message (`init`) is used in the initialization step to assign hop-count based addresses.
2) The `toSink` message is used to send a message from a sensor node to the data sink to perform a data storage operation.
3) The `fromSink` message is used to broadcast a message from the data sink server to every sensor node. This message carries the ID of the sending data sink. This type of message is used when the server proactively retrieves data from the sensors or when it needs to broadcast control messages to the sensors.
4) The `peer` message is used to communicate among the peer sensors.
5) The `nodeFail` message is used to inform nodes about a failed node. This type of message is used by successors of a failed node to negotiate new routing paths.

We first describe how the initial setup is performed, where one or more de Bruijn-style addresses are assigned to each sensor node. Then we illustrate how message routing is performed. Finally, we explain how resilience against node failures is achieved.

A. Initialization

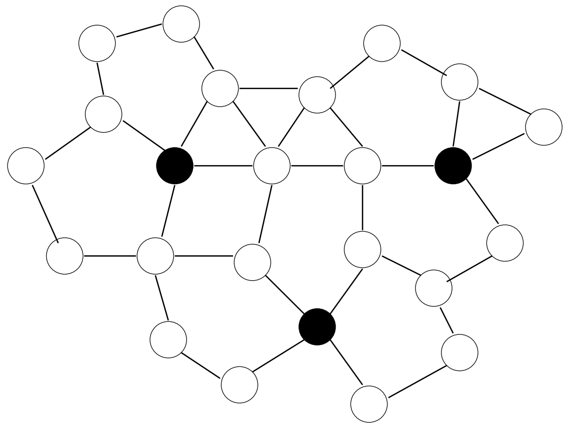
The data sink servers start the initialization step by a dynamic address assignment procedure. The t data sink servers have addresses $1, \ldots, t$. Suppose that the data sink server i has h sensors within its one-hop radio range. The data sink server i assigns the h sensor nodes the addresses $(i,0),(i,1),\ldots,(i,h-1)$. When a sensor node s with h' one-hop neighbors receives an address $\underline{a} = (a_1, a_2, \ldots, a_\ell)$ from an one-hop neighbor j, then it takes one of the following actions:

- If s does not have a valid address, then s takes $\underline{a}$ as its address. And it assigns each one-hop neighbor, except j, an address in the range of $(a_1, a_2, \ldots, a_\ell, 0), \ldots, (a_1, a_2, \ldots, a_\ell, h'-2)$.
- If s already has a valid address of length ℓ, then it keeps $\underline{a}$ as an alias address. Notice that all aliases of a sensor node have the same length.
- If s has a valid address of length $\ell' > \ell$, then it deletes all its address aliases and keeps $\underline{a}$ as a new address. And it once again assigns each one-hop neighbor, except j, an address in the range of $(a_1, a_2, \ldots, a_\ell, 0), \ldots, (a_1, a_2, \ldots, a_\ell, h'-2)$.

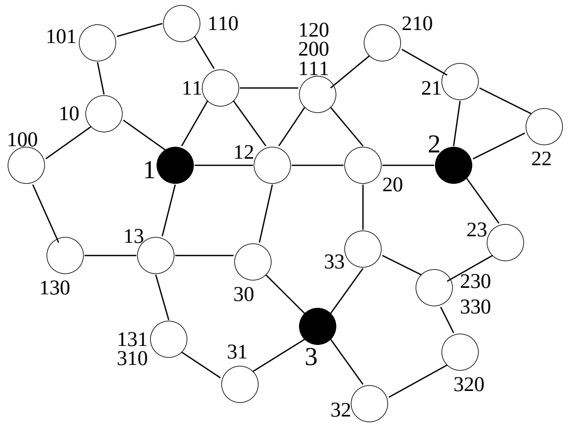
In this way, every sensor node that is reachable from a data sink will receive at least one address. The number of address aliases of a sensor node does not exceed the number of its one-hop neighbors. A sensor node informs its one-hop neighbors about its address aliases.

This simple address assignment scheme has some remarkable properties: If a sensor node has an address alias $(a_1, a_2, \ldots, a_\ell)$, then there is a path of $\ell - 1$ hops to the data sink a_1, and there is no shorter path to a_1. This assignment scheme realizes the partitioning into cells. If a node has only address aliases that start with a_1, then it is within the cell of a_1. The border nodes are characterized by the fact that they have address aliases that start with different digits.

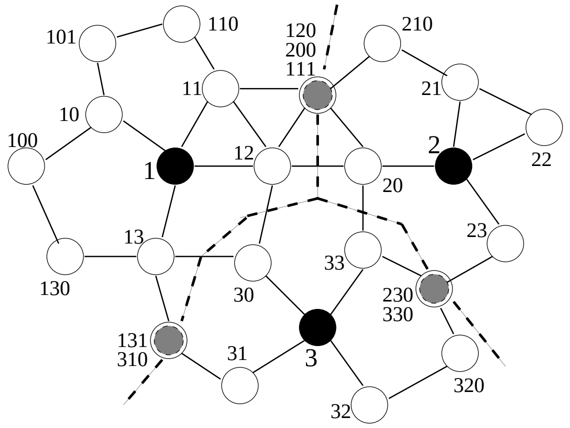
An example will be helpful to illustrate the main features of the address assignment. Figure 2 (a) shows a sensor network with three data sinks (that are depicted by black circles) and several sensor nodes (that are depicted by white circles). If two nodes are within radio-range of each other, then there is an edge between these nodes. The result after address assignment is shown in Figure 2 (b). Figure 2 (c) illustrates the subdivision into different cells. Each cell contains a data sink and all sensor nodes that are closer to this data sink than to any other in terms of hop-count. If a sensor node has the same distance from more than one data sink, then it belongs to the cell of



(a) A sensor network with three data sinks (black nodes) and several sensor nodes (white nodes); an edge between two nodes indicates that the nodes are within radio range.



(b) Sensor network after address assignment. Some nodes have several address aliases that lead to different routes in `toSink` messages.



(c) Induced partition of the network. All nodes that have an address alias beginning with the same digit belong to the same cell. Border nodes belonging to two different cells are shaded grey.

Fig. 2. Address assignment.

each of those data sinks; such nodes are called border nodes. The nodes 120, 230, 310 are examples of such border nodes.

If a sensor node s has address $(a_1, a_2, \ldots, a_{\ell-1}, a_\ell)$, then there exists a node p with address $(a_1, a_2, \ldots, a_{\ell-1})$. We call p a *predecessor* of s, and s a *successor* of p. The *associates* of s are all one-hop neighbors of s that are neither predecessors nor successors.

B. Routing

After the addresses have been assigned to the nodes, we can perform routing. The most common type of message is a `toSink` message from a sensor node to a data sink, which is typically routed through predecessors. Occasionally, a data sink may send `fromSink` messages to the sensor nodes, which are forwarded through successors. A `peer` message is routed through any combination of predecessors, associates, and successors.

- A `toSink` message is routed by randomly selecting one predecessor; this is done by right-shifting one randomly selected address alias. Then the same process is repeated until the data sink is reached. For instance, one possible path from the address $(a_1, \ldots, a_\ell)$ is through the predecessors
$$(a_1, \ldots, a_{\ell-1}), (a_1, \ldots, a_{\ell-2}), \ldots, (a_1, a_2)$$
to the data sink a_1.
- A `fromSink` message is broadcast by sending the message from the data sink to its successors, and each sensor node receiving such a message forwards it to all its successors.
- Suppose that a `peer` message is sent from a node with address $\underline{a} = (a_1, \ldots, a_\ell)$ to a node with address $\underline{b} = (b_1, \ldots, b_k)$. The node $\underline{a}$ or any node receiving the message forwards it to the one-hop neighbor that has an address alias with the longest common prefix with $\underline{b}$; if several one-hop neighbors qualify, then the one with the shortest address alias is chosen. If a data sink $\neq b_1$ receives such a message, then it will forward it to the data sink b_1.

We remark that the design of the protocol ensures that the routing of the `toSink` messages is optimal; in a typical sensor network application the `toSink` messages are by far the most frequent ones, since they are used to communicate the sensor data.

Example 1: Suppose that the node 131=310 in the sensor network given in Figure 2 (c) wants to send a `toSink` message to a data sink. If it chooses its alias 131, then the resulting route will be $131 \rightarrow 13 \rightarrow 1$. If it chooses its address alias 310, then the resulting route will be $310 \rightarrow 31 \rightarrow 3$.

Peer messages can be used, for example, by a sensor to check whether its sensor readings are reasonable. Although such messages are rare or not used at all in typical sensor network applications, we remark that routing between any two nodes is possible. Let us first look at an example that illustrates this routing rule.

Example 2: Consider the sensor network given in Figure 2 (c). Suppose that node 110 wants to send `peer` message to node 210. Since both neighbors of 110 have an empty common prefix with 210, the message is forwarded to 11, the shorter address alias. Among the neighbors of 11, the node 200 has the longest common prefix with 210, so it is routed there, and node 200 routes the message to 210.

Example 3: Suppose that node 130 wants to send a peer message to node 210 in the sensor network given in Figure 2 (c). Then the message is routed through $130 \rightarrow 13 \rightarrow 1$, then forwarded to data sink 2, and the final hops are $2 \rightarrow 21 \rightarrow 210$.

A straightforward routing rule for peer messages could use a sequence of predecessors until the node with the longest common prefix of $\underline{a}$ and $\underline{b}$ is reached, from which $\underline{b}$ can be reached through successors. Our `peer` message routing rule improves upon this rule by taking shortcuts whenever information about one-hop neighbors reveals such a possibility, as was shown in Example 1. Unlike `toSink` routing, it should be noted that `peer` routing is not necessarily optimal; this is the price one has to pay for the very limited memory usage. In view of the fact that `peer` messages are rare and typically local, this does not appear to be a significant disadvantage.

Remark. Our protocol makes typically multiple paths available while routing from sensor node to a data sink; unlike many other routing protocols for sensor networks, such as directed diffusion, ours will always ensure that the selected route is optimal, so that load balancing does not come at the cost of energy efficiency.

C. Fault-Tolerance

The failure of a node has significant impact on its successors and, to some extend, on its associates. Indeed, if a node fails, then its successor nodes might not be able to further deliver their sensor data to a data sink, unless some corrective measures are taken.

Fortunately, if a sensor node or a data sink fails, then this can be easily detected by a simple acknowledgment scheme. Therefore, we can assume that the one-hop neighbors of a failed node s become aware of the failure of s within a short amount of time. If a node s fails, then its one-hop neighbors take the following action:

- A predecessor of s informs the data sink that the node s has failed.
- All associates of s delete the address aliases that belong to s from their lists.
- All successors of s make their address aliases invalid that have an address alias of s as a prefix, and they send a `nodeFail`(s) message to their successors.

Each node receiving a `nodeFail`(s) message makes its address alias invalid that has s as a prefix, and forwards `nodeFail`(s) to its succcessors. Basically, the effect of the `nodeFail` messages is that all nodes that potentially route through s will eliminate this possibility.

Sensor Node Failure: Let us first illustrate the effect of a sensor node failure, and after that the failure of a data sink. Conceptually, the two concepts are the same, but a sensor node failure has much less impact in our scheme.

Recall that a sensor node might fail, for instance, because it lacks battery power. If a sensor node goes to sleep to save battery power, then we can also view this as a "temporary" failure; the only difference to a true failure is that the sensor node itself can inform its neighbors that it will become unavailable.

For example, let us assume that the sensor node 13 in Figure 2 fails. Figure 3 shows the situation right after the failure, when the neighboring nodes have detected that the node 13 is not available anymore. Figure 3 illustrates that

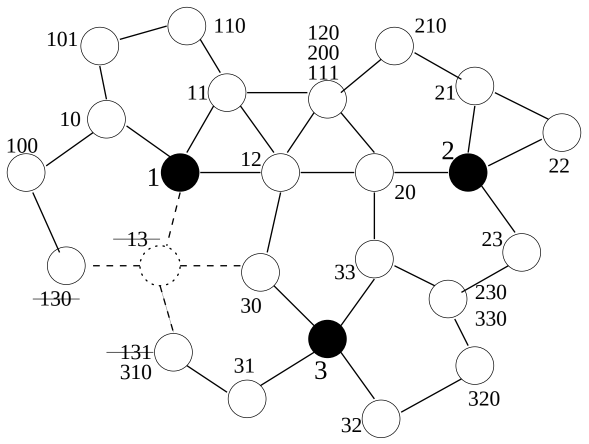


Fig. 3. Failure of sensor node 13.

the address aliases 130 and 131 are now invalid, since one cannot route through the predecessor 13 anymore. The node 131=310 is hardly affected, since it still can route through its predecessor 31. As there are no successors of 131, no further updates are necessary. On the other hand, the node 130 does not have any other address alias left after the failure of node 13, meaning that it might have to route its messages to the data sink through a longer route. By querying its neighbors (in this case, only node 100), it receives and accepts the new address alias 1000. Figure 4 shows the re-assigned address aliases after the recovery from the failure of node 13.

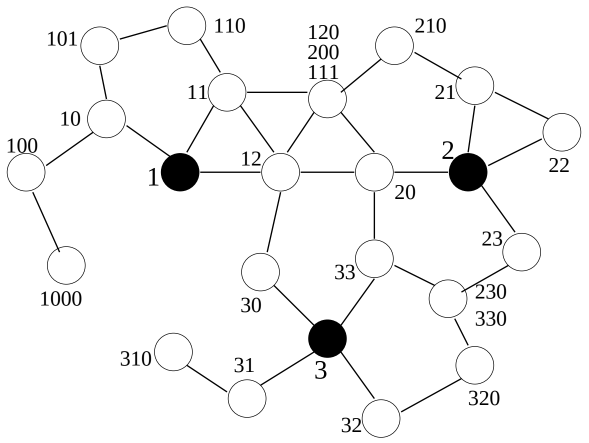


Fig. 4. Recovery after failure of sensor node 13.

It should be noted that if there is a sufficiently dense population of sensors so that each sensor has several address aliases, then the sleeping modes of the sensors will not cause much disruption of the routing. The successors (and their successors, ...) of a sensor node t that decides to go to sleep have to make their address aliases with prefix t invalid, but if there are other address aliases available, then this will not affect the routing.

Data Sink Failure: Data sink failures obviously have the biggest impact. For intance, let us assume that the data sink 3 in Figure 2 fails. The effect is that all addresses of sensor nodes within the cell of data sink 3 become invalid, with the exception of the border nodes, as shown in Figure 5.

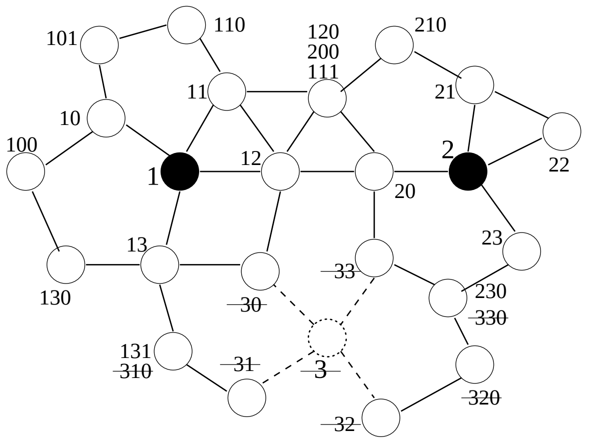


Fig. 5. Failure of data sink 3.

The recovery from a data sink failure works exactly in the same way as in the case of a sensor node failure: If a sensor node t does not have any valid address anymore, then it queries its one-hop neighbors to assign him address aliases. Then t proceeds exactly as in the initial address assignment step; it keeps only the shortest address aliases received, and rejects all others. Figure 6 illustrates the result of these address negotiations.

V. Experiments

Suppose that a sensor network consists of n sensor nodes. Let us assume that every sensor node and every data sink has about d one-hop neighbors, where $d \ll n$. Given a regular distribution of the sensor nodes, we can safely assume that the diameter of the sensor network is $O(\sqrt{n})$.

Let us define the *message complexity* of a sensor network as the number of messages that need to be delivered by the sensor nodes until a message sent from a source node reaches its final destination node.

The message complexity of a simple flooding scheme is $d \times n \times \Omega(\sqrt{n}) = \Omega(dn^{\frac{3}{2}})$. In contrast, in our scheme the message complexity for communication between a sensor node and a data sink is at most $O(\sqrt{n})$, and for `peer` messages the message complexity is at most $O(2\sqrt{n})$.

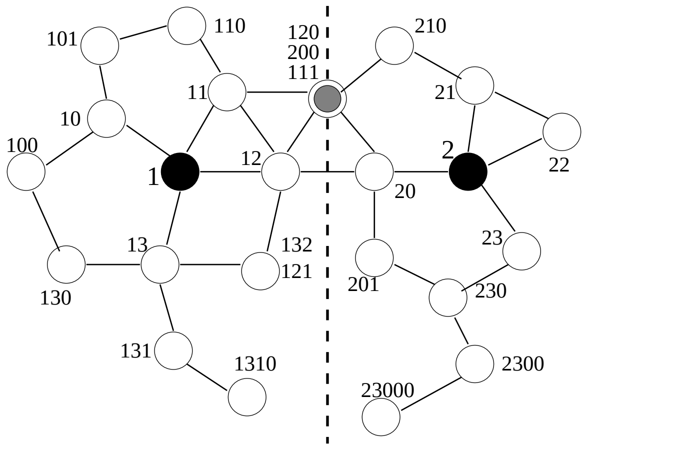


Fig. 6. Re-assignment of addresses after failure of data sink 3 and the induced partition into two cells.

We define the *fault tolerance* of a sensor network to be the probability of messages from any sensor node being successfully stored at any data sink. Therefore, it suffices to find the probability of a network partition.

We implemented a subset of our protocol to experiment with the expected behavior of a wireless sensor network. The topology we used as the input to the simulator is a $\sqrt{n} \times \sqrt{n}$ grid of n sensors and a single data sink in the middle. First, we measured the node connectivity by computing the expected number of (non-faulty) sensors that can still communicate with the data sink when k-out-of-n sensors fail, which is shown in Figure 7.

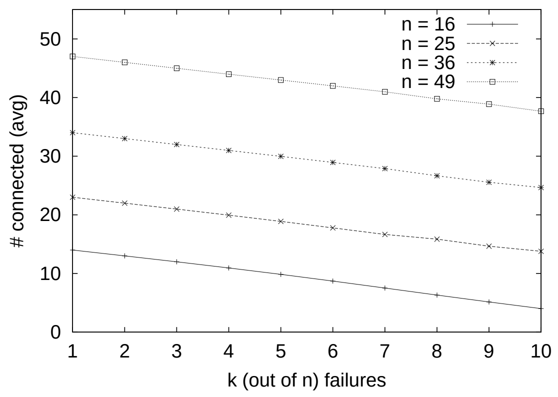


Fig. 7. Expected node connectivity.

In a second experiment, we simulated the expected number of hops taken by `toSink` messages. The result is shown in Figure 8. The plot shows the average value of $\binom{n}{k}$ different scenarios of k-out-of-n failing nodes.

The simulation results show that in the case of node failures our scheme is able to maintain the connectivity between the nodes and the data sink. At the same time, it maintains a short average path length, as expected.

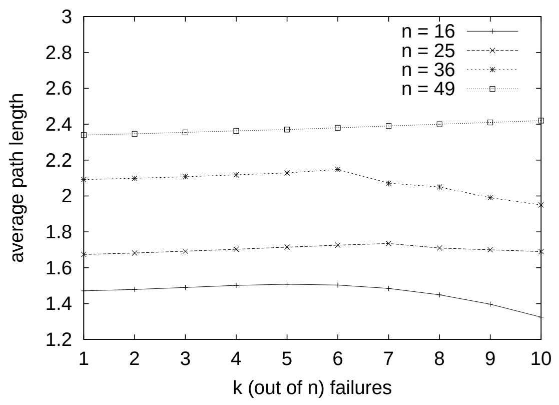


Fig. 8. Average path length.

VI. Conclusions

We proposed an energy-efficient communication protocol for data storage and retrieval in a wireless sensor network. Our protocol employs replicated data sinks to improve fault-tolerance in the face of data sink failures. We achieve resilience against sensor node and data sink failures through a dynamic re-assignment of addresses and the introduction of alternate paths.

The most common application in sensor networks is the delivery of sensor data. Our protocol ensures that such messages from the sensor nodes are always routed in an optimal way to the closest data sink using the least possible number of hops. We avoid the overhead of keeping routing tables to accomodate the memory constraints of sensor nodes; a node simply needs to keep the address aliases of itself and of its one-hop neighbors. Furthermore, our protocol does not require any location information. The reasonably low message complexity of our scheme can extend the battery life of each node, maximizing the overall life of the sensor network.

Acknowledgments

The research by H.L. was supported by the University of Denver PROF grant 88197. The research by A.K. was supported by NSF grant CCR-0218582 and NSF CAREER award CCF-0347310.

References

[1] B. Bollobás, *Extremal Graph Theory with Emphasis on Probabilistic Methods*. Providence, RI: American Mathematical Society, 1986.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry*. Germany: Springer-Verlag, 1997.

[3] M. Demirbas, A. Arora, and V. Mittal, "FLOC: A fast local clustering service for wireless sensor networks," *Workshop on Dependability Issues in Wireless Ad Hoc Networks and Sensor Networks (DIWANS/DSN)*, 2004.

[4] C. Gui and P. Mohapatra, "SHORT: self-healing and optimizing routing techniques for mobile ad hoc networks," *Proc. of the 4th ACM Int. Symp on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pp. 279–290, June 2003.

[5] M. Handy, F. Grassert, and D. Timmermann, "DCP: A new data collection protocol for Bluetooth-based sensor networks," *EUROMICRO Symposium on Digital System Design*, pp. 566–573, 2004.

[6] D. Hsu and D. Wei, "Efficient routing and sorting schemes for de Bruijn networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 11, pp. 1157–1170, November 1997.

[7] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," *Proc. of the 6th Annual Int. Conf. on Mobile Computing and Networking (MobiCom)*, pp. 56–67, August 2000.

[8] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table," *Proc. of Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[9] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," *Proc. of the 6th Annual Int. Conf. on Mobile Computing and Networking (MobiCom)*, pp. 243–254, August 2000.

[10] M. Kochhal, L. Schwiebert, and S. Gupta, "Role-based hierarchical self organization for wireless ad hoc sensor networks," *Proc. of the 2nd ACM Int. Conf. on Wireless Sensor Networks and Applications (WSNA)*, pp. 98–107, September 2003.

[11] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," *Proc. of the 5th Symp. on Operating Systems Design and Implementation (OSDI)*, 2002.

[12] N. Sadagopan, B. Krishnamachari, and A. Helmy, "The ACQUIRE mechanism for efficient querying in sensor networks," *Proc. of the First IEEE Int. Workshop on Sensor Network Protocols and Applications (SNPA)*, May 2003.

[13] M. Samatham and D. Pradhan, "The de Bruijn multiprocessor network: a versatile parallel processing and sorting network for VLSI," *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 567–581, 1989.

[14] A. Samsudin and K. Y. Lee, "nD-dBPN: New self-routing permutation networks based on the de Bruijn digraphs," *Proc. of the 1998 Int. Conf. on Parallel Processing*, pp. 604–611, August 1998.

[15] R. C. Shah, S. Roy, S. Jain, and W. Brunette, "Data MULEs: Modeling a three-tier architecture for sparse sensor networks," *Proc. of the First IEEE Int. Workshop on Sensor Network Protocols and Applications (SNPA)*, May 2003.

[16] M. Shooman, *Reliability of Computer Systems and Networks*. New York: Wiley, 2002.

[17] K. N. Sivarajan and R. Ramaswami, "Lightwave networks based on de Bruijn graphs," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, February 1994.