Efficient Data-Centric Storage in Sensor Networks with Lightweight Routing and Address allocation algorithm

Ghazi AL SUKKAR
Institut National des Télécommunications
Evry-France
ghazi.al_sukkar@int-evry.fr

Hossam AFIFI
Institut National des Télécommunications
Evry-France
hossam.afifi@int-evry.fr

Sidi-Mohammed Senouci
France Telecom R&D
Lannion-France
sidimohammed.senouci@francetelecom.com

Abstract:
In this paper we propose two algorithms for efficient data-centric storage in wireless sensor networks without the support of any location information system. These algorithms are intended to be applied in environments with large number of sensors where the scalability of the network has great issue. During the first algorithm, each sensor obtains a unique temporary address according to its current relative location in the network. The second algorithm is used to route data from one sensor to another, this routing algorithm only depends on the sensor's neighborhood, i.e. in order to implement the routing table each sensor needs only to exchange local information with its first hop neighbors. The forwarding process used in this algorithm resembles the one found in Pastry peer-to-peer protocol.
Our simulation results show that these algorithms scale well with network size and density.

Keywords: Sensor networks, Data-centric storage, Routing, Distributed Hash tables, Dynamic Address Allocation.

I. Introduction:

Sensor networks are expected to play an important role in future communications, where they will find wide application scenarios in daily life events. Large-scale events such as disaster relief or rescue efforts are highly dependent on effective communication capabilities.
Wireless sensor networks facilitate monitoring and controlling of physical environments from remote locations with better accuracy. A wireless sensor network consists of a large number of miniature sensors, mostly with scarce resources, (e.g. memory, communication range, processing power, and most importantly, battery power), which collect and disseminate environmental data.
A wide range of application scenarios are proposed in the literature for wireless sensor networks [1, 2, 3, 4], examples of such applications are: safety monitoring, real-time pollution monitoring, wildlife monitoring, military sensing and tracking, etc.
For disseminating and storing the sensed data, three methods are available in the literature [5, 6]:

1. Local storage: here, each sensor keeps the data it senses locally. To retrieve data, a query must be flooded through the network, causing sensors with data relevant to the query to send data back to the base station.
2. External storage: in this method, data is sent to the base station without waiting for a user to send a query. While external storage avoids flooding the network with a query, it may waste energy when data that the user is not interested in is sent to the base station.
3. Data-centric storage: in data-centric storage, events are named, and sensors cooperate locally to detect named events. When a sensor detects a named event, it determines which sensor is responsible for that name, and then stores the data at that sensor. Which sensor is responsible for storing a type of data is typically determined by taking a hash of the name, and mapping that hash onto a sensor in the network. When a user wishes to query the network, he can send the query only to the sensor responsible for the data relevant to the query. Note that in this approach, queries do not need to be flooded through the network, nor does data that the user does not ask about get sent to the base station. Additionally, the query may be partially processed at the sensors storing the data, allowing a small message consisting of aggregated data to be sent to the base station instead of all individual records relevant to the query.

Data-centric storage provides a *(key, value)* based associative memory, in a way similar to the distributed hash table (DHT) systems designed for the internet use, like Pastry [7], CAN [8], Chord [9], and Tapestry [10] where nodes communicate in an application level fashion through the formation an overlay network between them.
In data-centric storage, events are named with keys and both the storage of an event and its retrieval are performed using these keys. Thus the two operations available in data-centric storage based sensor network are:
Put(k,v): which stores the observed data v according to the key k.
Get(k): retrieves whatever stored value associated with key k.

As shown by [6] data-centric storage is preferable in cases where (a) the sensor network size is large, (b) there are many detected events and not all event types are queried. In this paper we will concentrate on this method, since it seems to be the most efficient way of data dissemination and storage in sensor networks.
Such sensor networks are supposed to be unsupported by an underlying IP infrastructure and independent of the IP-like hierarchical addressing.

In this paper, we present two correlated algorithms for efficient data-centric in sensor networks. They are completely distributed algorithms without any centralized control, which result in all sensors have identical responsibilities.
In the first algorithm a sensor is assigned a unique address according to its relative location in the network, the address assignment mechanism works in a distributed manner where address conflict is avoided without the need to flood the whole network. Sensors change their addresses as they move, so that their addresses have a topological meaning.
The second algorithm is the routing algorithm which is very simple and depends only on the node's first hop neighbors, and the forwarding process resembles to some degree the one found in Pastry peer-to-peer protocol [7].

The rest of this paper is organized as follows. In section II we describe the related work; the framework design requirements are discussed in section III. In section IV we describe the address allocation procedure, section V describes the routing procedure, data centric storage mechanism is described in section VI, topology dynamism and address reassignment are explained in section VII, performance analysis and comparison are treated in section VIII. Finally we conclude with section IX.

II. Related Work:

Ratnasamy et al. [5] were the primaries who introduce the concept of data-centric storage in sensor networks, as well as they describe the simpler concepts of local and external storage. Their work continued in [6], where they describe GHT, a data-centric storage system for sensor networks built on top of GPSR[11]. Their approach depends on the use of geographical information where they assumed that each node knows its location coordinates using some technologies (e.g. GPS), although this works well, however, location information is not always available, current methods of determining the location information consume much energy and may not be possible in many sensor network scenarios.
Also GPSR works best when geographic locality accurately represents network topology. For many sensor networks, geographic locality may differ significantly from network topology i.e. physical obstacles can easily prevent two geographically close nodes from communicating directly, causing them to be far apart in the network topology.

Taking this in consideration, a number of new routing protocols where invented, that try to estimate node coordinates in a relative way without the assistant of any positioning system, examples of these protocols NoGeo [12] and GEM [13]. NoGeo proposed an algorithm for performing node to node routing with only neighbor information, without geographic location information. Where a virtual coordinate system is built by having nodes on the perimeter of the network determine their positions relative to each other. They then use an iterative relaxation algorithm for other nodes to determine their coordinates. Once the coordinate system is built, they use greedy forwarding like the one in [11] to perform all the routing. This approach has the advantage that node failures and node mobility are more easily dealt with, also this approach is interesting, in that it achieves the *O(1)* complexity of geographical routing. However, there is a relatively large set-up overhead for the perimeter nodes to find their relative positions, and to perform several iterations of the relaxation algorithm. Also the scheme will only work on certain types of graphs (typically unit-disk like graphs).

GEM constructed a labeled graph that embedded in the original network topology in a distributed fashion. In that graph, each node is given a label that encodes its position in the original network topology.
To do that they developed two algorithms, with the first one VPCS, they embedded a ringed tree into the network topology, and labeled the sensors in such a manner as to create a virtual polar coordinate space. They have also developed VPCR, a routing algorithm where each node keeps state only about its immediate neighbors, and requires no geographical information.
While they provide a simple routing algorithm within the virtual polar coordinate space, the overhead to implement this virtual coordinate is considered to by high.
Where in order to use the routing algorithm efficiently they resort to align the virtual space with the network topology using two techniques, in the first scheme called the Naïve scheme, sensors are required to fine the distances between themselves and their neighbors by observing the radio signal attenuation or by measuring the arrival time difference between radio and ultrasonic pulses. Even doing this will not guarantee a good performance, since small errors in distance estimation prevent a good aligning in the virtual space.
In the second scheme each sensor calculates its position from knowing its distance in number of hops to at least three reference nodes including the root node, and the distances between these references.
To calculate these distances, they build a spanning tree from each of the reference nodes. Every node then knows its distance in network hops from each reference node. One of the reference nodes can then send the distance between itself and the other reference node to the root node. The root node then floods the network with the distances in network hops between itself and the two other reference nodes.
Since the distances are measured in number of hops, the resulting position estimation are not highly accurate, to solve this problem they resort to another method called centers of mass which result in a relatively high overhead.
Several protocols for wireless sensor networks were proposed, which can be classified under the Local storage method, such as LEACH[14] and Minimum Cost Forwarding[15].
Moreover, Directed diffusion[16] and TAG[17] are more advanced forms of external storage.

III. Framework design requirements:

Topology dynamisms, scare resources, failure of nodes, and scalability of the network are challenging issues in any design for a data-centric storage system in sensor networks. Thus a good framework design for data-centric storage should guarantee the following requirements [6]:

- **Persistency:** a (k, v) pair stored in the system must remain available to queriers, despite sensor node failures and changes in the sensor network topology.
- **Consistency:** a query for k must be routed correctly to a node where (k, v) pairs are currently stored; if this node changes (*e.g.*, to maintain persistence after a node failure), queries and stored data must choose a new node consistently.
- **Scaling with network size:** as the number of nodes in the system increases, the system's total storage capacity should increase, and the communication cost of the system should not grow unduly. Nor should any node become a concentration point of communication.

As we will observe, our proposed algorithms try to pursue these requirements in an efficient way.
In our algorithms a sensor is dynamically assigned a unique address which changes with its movement to reflect sensor's location in the network, this address is used to simplify routing in the network.
To join the network, a sensor establishes a physical connection to at least one node already in the network and requests an address. The neighbor node(s) answer(s) with an address. As a sensor moves, it requests and receives new addresses from its new neighbors.

The forwarding is done in a way similar to the one done in Pastry [7], one hop at a time, where each node forwards the message to its immediate neighbor who gets the message as close as possible to the destination.

IV. Address Allocation Algorithm:

This algorithm enables the sensors to allocate addresses in a local way i.e. without the need to contact faraway nodes in the network or flooding the whole network, where at any given time; each node manages a range of addresses including its own address. Node addresses are dynamically assigned depending on the node's current position in the network. More specifically, the addresses are organized as a tree. We call this the *address tree*, see Fig. 1.

To understand this addressing assignment mechanism, let us assume that the addresses are *d* digits with base 10 numbers, so addresses will be in this form $A_{d-1}, \ldots, A_0$, where $A_i \in \{0,1,...,9\}$. As we will notice, the choice of the base B will determine the maximum number of children a node could have, in our example here the maximum number of children is 9.

The base station will be a logical choice to play the role of the first node which will form the network (since it is the most stable node), so it will take the all zeroes address 00. . .0, we call it the root node, as sensor nodes arrive[1] in the neighborhood of the root (i.e. they are in its transmission range), they contact it to obtain an address (call these nodes level 1 nodes). The root node control the first digit (leftmost digit) of the address, where it give the first arriving node address 100...0, the second arriving node 200...0 and so on up to 900...0. These first level nodes control the second digit (from left) in the address, so when nodes connect to any of these nodes and ask for address, they fix the first digit as their address and change the second digit according to node arriving sequence. For example if a node arrive and it is in the neighborhood of the node with address 100...0 and ask this node for an address, then node 100...0 will give it the address 110...0, the second node ask 100...0 for an address will take 120...0 as an address and so on (we call node 100...0 parent of nodes 110...0, 120...0,...,190...0 and thus they are its children).

These second level nodes take control of the third digit and so on. Fig. 1 show an example of an address tree with three digits addresses, for $d = 3$ digits, the entire address space can be represented by *xxx*, where $x \in \{0, 1,..., 9\}$, nodes in level *l* subtree are the children of the node in level *l-1*. We call the last level nodes in the tree *leaves*.

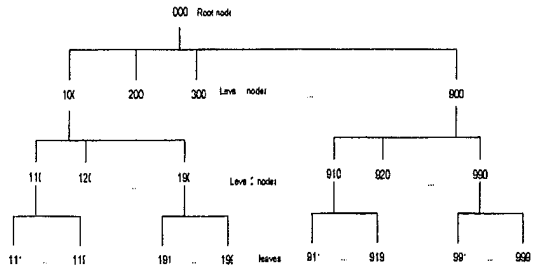

Fig. 1. Address tree with three digits decimal address space.

[1] Assume nodes arrive in the network one by one.

These leaves do not take control of addresses since address space reaches its limit.

Address tree illustrates how addresses are allocated; it does not represent the actual network topology although address of a node depends on its current position in the network. Fig. 2 shows an example of a network topology which uses this algorithm.

When a new node *i* arrives in the network, it receives an address R_i (call it temporary address) which will be used for routing. A new node in the network receives the temporary address from one of its neighbors (we call this neighbor the *parent neighbor* P_i). We assume the existence of some bootstrap mechanism which allows new nodes to identify their neighbors in the network.

This mechanism results in a list containing information about all neighbors. Let $N_i = \{n_1, n_2, ..., n_q\}$ be the set of q nodes in the neighborhood of node *i* (in its transmission region). The neighborhood list L_i of node *i* is defined as

$$L_i = \left\{ \left(n_1, R_{n_1}, C_{n_1} \right), \left(n_2, R_{n_2}, C_{n_2} \right), \ldots, \left(n_q, R_{n_q}, C_{n_q} \right) \right\}$$

where $C_{n_j} = \{R_{c_1}, R_{c_2}, \ldots, R_{c_m}\}$ is the *children list* managed by node n_j, $\forall\, n_j \in N_i$.

The neighborhood list is used to determine which existing node in the neighborhood will give a temporary address to the arriving node. Several factors must be taken into account.

We apply the following criteria to assign one temporary address to a new node. Using this criterion the joining node selects, among a set of candidate neighbors, the node which will be the parent neighbor of it. This node will be the one with the least level i.e. the nearer to the root. If two or more nodes have the same level then it chooses the node with the least number of children, if a gain two or more nodes satisfy this condition then it will choose the one with the least address.

After the new arriving node chooses the parent neighbor it asks that parent for a temporary address which will be assigned according to our address allocation algorithm, we said that an *association relationship* established between the two nodes. In Fig. 2 this association relationship is represented by continuous thick lines, where the dotted thin lines represent the *neighborhood relationship*.

V. Routing Algorithm:

The previously mention address allocation algorithm simplifies the routing procedure, as we will see. Where routing is performed in a hop by hop basis.

Having obtained its temporary address, the new node *i* also learns the temporary addresses of its immediate neighbors. This neighborhood information will compose its routing table.

In this algorithm a node routes a message by simply forwarding to the neighbor whose address is the closest to the searched temporary address of the destination until the messages reaches the destination. This forwarding procedure resembles the forwarding procedure in Pastry [7]; where the message is forwarded to a node from the routing table that has a temporary address with longer shared prefix with the temporary address of the destination.

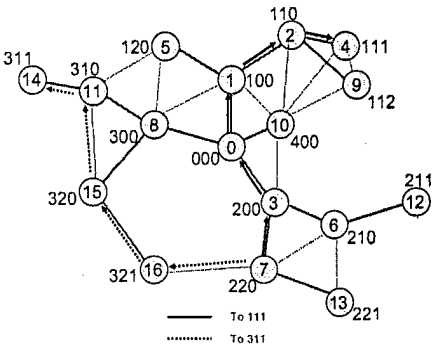

Fig. 2. An example of network topology with 17 nodes and three digits address space. Numbers in the circles are nodes identifiers and at the same represent the sequence of nodes arrival at the network; numbers beside the circles are nodes addresses.

If the node can not find in its routing table such a node that have a longer shared prefix matching, it simply forward the message to its parent an so on until the message reach its destination.

Fig. 2 shows an example of how the routing algorithm works, here node 7 with $R_7 = 220$ want to sent for the destination 14 with $R_{14} = 311$. Node 7 find in its routing table that node 16 has a temporary address that matches the destination temporary address in the first digit, so it forwards the message to this neighbor, in its turn node 16 forwards this message to node 15 which is its parent neighbor since it does not have in it routing table any node that has a longer prefix matching with the destination node's temporary address. Node 15 forward the message to node 11 which has a temporary address that matches the destination's temporary address in two digits. Finally, this node forwards the message to node 14 which is the destination node.

Also Fig. 2, illustrates another routing example, where the source is node 7 and the destination is node 4. As you can note from this example, the message forwarded back to the root node 0 which in its tern forward it to the destination.

The arrival of a new node affects only a limited number of existing nodes (nodes that are in its direct transmission region). The number of neighbors and, consequently, the signaling overhead, depend only on the node's transmission range and are independent of the total number of nodes in the system. Furthermore, a small amount of information suffices to implement this routing algorithm. Each node only stores information about itself and about its neighbors.

VI. Data-centric storage mechanism:

As we will see, implement these algorithms in sensor networks will simplify applying data-centric storage in these kinds of networks. So here we will explain how this is done.

1. Event Storing Procedure **Put**(k,v):

This operation is used to identify the node which will be responsible for storing a sensed named event v. We will assume the existence of previously known naming system, which maps each defined event to a key k.

By using any well-known functions like SHA-1 [18], the sensor i which detect the event v, hashes the key of the sensed event, and obtains an m-bit number. This number is then translated using certain function into another number which falls in the temporary address space, this number R_r is used to find the sensor which will be responsible of storing the event data v, as the following.

Sensor i forwards a *registration* message using R_r as a destination address, by applying the routing procedure as in section V. This request will be forwarded until it reaches the sensor having temporary address that has the longest prefix matching with R_r.

So this sensor is the one responsible for storing the sensed data event v.

2. Event lookup Procedure **Get**(k):

The interested node s apply the same globally know hash function on the events key, so it well get a temporary address R_d, this temporary address is the one used to find the sensor which is responsible for storing this event value v.

To find this sensor, the node s forwards a *lookup* message using R_d as a destination address, applying the routing algorithm in section V, this message will be forwarded until it reaches the sensor with the longest prefix matching with R_d, this sensor is the final destination. So it is our target, which will respond with the value v corresponding to the key k.

VII. Topology dynamism and Address Reassignment:

Our algorithms have to deal with dynamic topology changes caused by sensors voluntarily join or leave the network due to sensors mobility, or by sensors failures due to energy depletion (though some may fall prey to a hostile environment). When a sensor i departure, the system must guarantee the stability of the routing protocol, also the persistency and the consistency requirements of the framework have to be guaranteed (see section III).

A. Persistency:

To ensure the persistency of the system in case of dynamic topology changes, the sensed event data v has to be stored in multiple locations. These locations have to be chosen in an efficient way.

We suggest here to store v in different branches of the address tree, so in case of a complete distortion in the first level subtree, the data v is guaranteed to be available in another subtree.

To do that we have to modify the hash function in such away to give use multiple numbers which will be used to store the data in multiple locations. One simple way to do that is to modify the left most digit in the number which result from applying the original hash function as in section VI.1:

B. Consistency:

To ensure consistency we have to deal with sensor movement and sensor sudden failure.

Dealing with sensor movement:

We consider that before leaving its location, a sensor explicitly hands over its temporary address R_i, its neighborhood set N_i, neighborhood list L_i, its children list C_i, and the associated mapping information database to its parent neighbor[1].

[1] We assume the existence of a mechanism that allows a node to determine when it is leaving its location.

In this situation we have to deal with one of the following two cases:

Case 1: The leaving sensor *i* is a leaf node, Fig. 3, shows an example, where node 9 leaves the network (or changes its position), in this case, the node mobility will cause no impact on the organization of the topology, the only process that will take place is the handover of the mapping information database, to the parent P_i, and the temporary address of the leaving node will be available again for its parent to be assigned to another node.

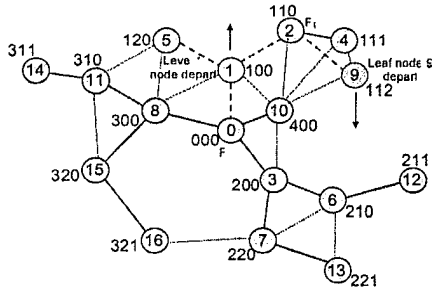

Fig. 3. Leaf node 9, and node 1 leaves (or changes its position).

Case 2: The leaving node is not a leaf node; it could be any node in any level of the address tree. So the system must guarantee the persistency and the consistency after a node departure. In Fig. 3, node 1 leaves the network, the parent neighbor is node 0, and it leaves behind four descendents; its two children, node 2 and 5, and the children of node 2; which are node 4 and 9.

Based on the received neighborhood list L_i of the depart node *i*, its parent neighbor P_i will face one of the following:

- The children of the leaved node *i* are also neighbors of the parent node P_i of *i* i.e. $C_i \subset N_{P_i}$, in this case the parent neighbor establishes an association relationship with these node, telling them that it now play the role of their previous parent *i* and no other operation will be required. Thus any messaged directed to (through) or from these children will be processed by the parent neighbor of the previously departed node. We call this a smooth reassignment.
- All or some children of the leaved node *i* are note neighbors of the parent node P_i of *i* i.e. $C_i \not\subset N_{P_i}$, in this case, the parent neighbor P_i try to find the set *S* of the children nodes that are also neighbors to it self, i.e. $S = C_i \cap N_{P_i}$, if it is not empty $S \neq \phi$, then the parent neighbor establishes an association relationship with these node as in the previous case, so these node will keep their temporary address.

For the rest of children $f = C_i - S$ that are note neighbors of P_i, after detecting the absence of their parent neighbor *i*, they will try to rejoin the network as they are a new arriving nodes, so they will ask another node with a higher level for a new temporary address and establish an association relationship with it, and inform all of its children nodes about the address change, so these nodes will change there addresses according to the new address of their parent, these children will do the same process, this process will repeated recursively until the change include all the nodes in their subtree.

If a node from set *f* could not find a node with higher level to establish an association relationship with it, this node will try to establish an association relationship with nodes in the same level as its current one or even with lower level.

This node can not be one from the subtree of the leaved node, in case that they still hold the previous temporary address.

At the same time this node will send a message for all of its children telling them to rejoin the network, in this case each one of these children will try in its turn to apply the same previous mechanism which will recursively repeated. See Fig. 4.

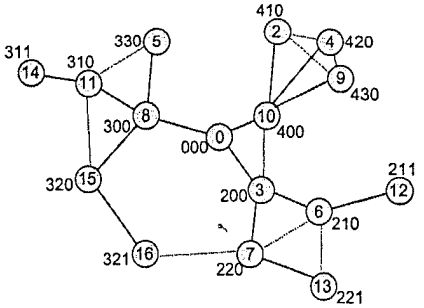

Fig. 4. This figure shows the network after node 1 leaved the network.

Taking this case in consideration, each time that a new node arrives in a location that has been previously occupied by another node, the parent neighbor verifies if the new node is appropriate to receive the previous handed over temporary address and the associated mapping information database.

For doing this, the parent neighbor compares the neighborhood set sent by the previous mobile node, before it's moving, with the one for the new arrived node. If the new node is also a neighbor of the children of the previously leaved node i.e. if $S \subset N_{NEW}$, the parent neighbor assigns the temporary address and the mapping information database of the previously leaved node to the new node. However, if the new node can not satisfy this condition a new temporary address will be attributed to it, according to the described joining procedure.

Dealing with sensor sudden failure:

The mechanism to deal with sensor sudden failure is same as the one used in sensor movement. The difference here is that in case of sensor sudden failure, its neighborhood set N_i, neighborhood list L_i, and its children list C_i of the failed sensor *i* are not available for its parent neighbor. In this case its parent neighbor and its children will depend on the received hello messages and the routing table, to decide if they are still neighbors, so they could apply the same mechanism in the case of sensor movement. This will result just in a higher handover time than the case of sensor movement, since discovering sensor sudden failure, and the dependency on the hello messages, will take longer time.

VIII. Performance Comparison and Analysis:

The scalability of these algorithms comes from the following features:

- Size of the routing table: where each sensor has a routing table of size $O(q)$, where q is the number of immediate neighbors of the sensor node.
- Signaling traffic needed to implement and maintain the routing table: the routing table entries are the immediate neighbors, and the only signaling traffic needed is the hello

signals between neighbors that used to inform that the sensor is still alive and still in its position.

- The arrival of a new sensor and sensor movement affects only a limited number of existing nodes (nodes that are in its direct transmission region). Thus the signaling overhead resulting from this action will be small and local.
- The cost of event storing is *O(1)*, since the only thing that a source needs to store event data is to route them to the destination using the number resulting from hashing the event key as a destination address.
- The cost of event lookup is also *O(1)*, since the source needs only to route the lookup message to the temporary address resulting from applying the event's key to the globally known hash function.

Our algorithms are almost similar to those found at GEM [13], but they outperform them in the following aspects:

- There is no need to align the virtual space with the network topology, which results in a high processing and communication overhead as we discussed before in section II.
- There is no need to assign the node level in the spanning tree with its label explicitly, since in our algorithms the node level is included implicitly in its label.
- No need to know the size (number of sensors) in each subtree of the spanning tree in order to distribute the polar coordinate space between nodes in a way proportional to this size.
- A new joining sensor in GEM, will affect a number of sensors already exist in the networks. In order to do this, the new joining node first chooses a parent from its set of neighbors. The parent assigns the new node a level equal to its own plus one. The parent then assigns the new node an angle range by first taking away part of the angle range from one of its other children. That child must take away that angle range from its child that it is assigned to as well. Thus, this change recurs down the tree to a leaf node.
- GEM use a two hops neighborhood information, while in our algorithm we use only one hop neighborhood information, these will reduce the amount of memory uses since it is one of scare resources in sensor networks.
- Node failure in GEM may result in a discontinuity in the angle range of some nodes, which makes the storage of neighbor angle range more complex.

IX. Conclusion:

Two algorithms were proposed for efficient data-centric storage in wireless sensor networks without the support of any location information system.

A small amount of information suffices to implement the routing table, *i.e.*, low signaling overhead is generated (only local neighborhood communication), Thus the routing table size is *O(q)*, where *q* is the number of immediate neighbors of the node.

We expect these algorithms to be applied in environments with large number of sensors where the scalability of the network has great issue.

Our future study will include sudden node failures and a treatment of certain network issues, like network separation, networks merging. And a study of this protocol performance through simulation. We are now in the state of implementing this protocol in NS2.

REFERENCES:

[1] D. Estrin, L. Girod, G. Pottie, M. Srivastava, *Instrumenting the world with wireless sensor networks*, International Conference on Acoustics, Speech and Signal Processing, Salt Lake City, UT, May 2001.

[2] J. Warrior, *Smart sensor networks of the future*, Sensors Magazine, March 1997

[3] G. J. Pottie, W. J. Kaiser, *Wireless integrated network sensors*, Communications of the ACM 43 (5) 2000, pp. 551-558.

[4] Cerpa, et al., *Habiatat Monitoring: Application Driver for wireless communication technology*, Workshop on Data Communications, ACM SIGCOMM, April 2001.

[5] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric storage in sensomets, *Proc. ACM SIGCOMM Workshop on Hot Topics In Networks*, 2002.

[6] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: a geographic hash table for data-centric storage, *Proceedings of the ACM Workshop on Sensor Networks and Applications*, pp. 78--87, Atlanta, Georgia, USA:ACM, September 2002.

[7] Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," *in Proceedings of the Middleware*, 2001.

[8] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker. "A Scalable Content-Addressable Network," In *Proceedings of the ACM SIGCOMM*, 2001.

[9] Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A Scalable Peer-topeer Lookup Service for Internet Applications," *ACM SIGCOMM 2001*, San Diego, CA, August 2001.

[10] Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in communications*, vol. 22, no. 1, pp. 41–53, January 2004.

[11] Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. Proceedings of the 6th annual international conference on Mobile computing and networking, Boston, Massachusetts, United States, 2000, pages 243–254.

[12] Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *ACM MobiCom*, 2003.

[13] James Newsome and Dawn Song, "Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, New York, NY, USA, 2003, pp. 76–88, ACM Press.

[14] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks, Proceedings of the 33rd Annual Hawaii International Conference on system sciences, 2000.

[15] F. Ye, A. Chen, S. Lu, and L. Zhang. A scalable solution to minimum cost forwarding in large sensor network, in Tenth International Conference on Computer Communications and Networks, 2001, pp. 304–309.

[16] Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. pages 56–67, Proceedings of the 6th annual international conference on Mobile computing and networking, Boston, Massachusetts, United States, 2000, pp. 56 – 67.

[17] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad hoc sensor networks, In *Proc. 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, 2002, pages 131–146.

[18] "FIPS 180-1, Secure Hash Standard." *U.S. Department of commerce/NIST, National Technical Information Service*, Springfield, Apr. 1995.