Lightweight and Distributed Algorithms for Efficient Data-Centric Storage in Sensor Networks

Ghazi AL SUKKAR, Hossam AFIFI
Institut National des Telecommunications
Evry, France

Sidi-Mohammed SENOUCI
France Telecom R&D
Lannion, France

***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.**

I. INTRODUCTION

A wide range of application scenarios are proposed in the literature for wireless sensor networks [1, 2], 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 [3, 4]:

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 [5], CAN [6], and Chord [7], 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 [5] data-centric storage is preferable in cases where (1) the sensor network size is large, (2) 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.

Here, 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 [5].

The rest of this paper is organized as follows. In section II we describe framework design requirements. The address allocation algorithm is discussed in section III. In section IV we describe the routing algorithm; section V describes data-centric storage mechanism. We conclude with section VI.

II. 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 [4]:

- **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 [5], 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.

III. 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 address is 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 $A_{d-1}, \ . \ . \ . \ , A_0$, 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 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 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*.

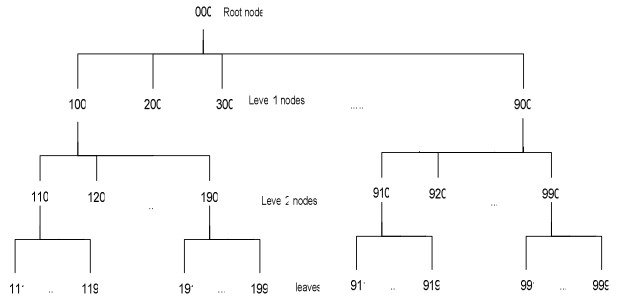

Figure 1. Address tree with three digits decimal address space.

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 (call it the *parent neighbor* P_i). We assume the existence of some bootstrap mechanism which allows new nodes to identify their neighbors in the network. 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*.

IV. Routing Algorithm

The previously mention address allocation algorithm simplifies the routing procedure, as we will see.

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 [5]; 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.

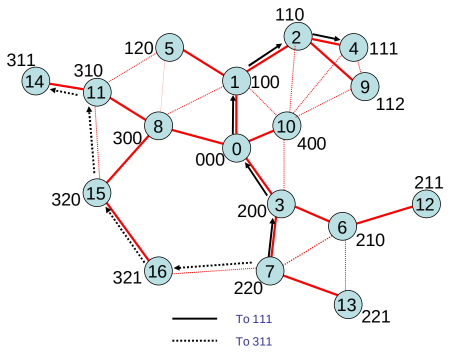

Figure 2. Numbers in the circles are nodes identifiers at the same timerepresent the sequence of nodes arrival at the network; numbers beside the circles are nodes addresses.

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.

Figure 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.

V. 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.

A. *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 [8], 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 IV. 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*.

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

The interested node *s* apply the same globally known hash function on the events key, so it will 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 IV, 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*.

VI. Conclusion

Two algorithms were proposed 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.

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.

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

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

[4] 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.

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

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

[7] 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.

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