Fault-tolerant and Load Balancing Localization of Services in Wireless Sensor Networks

Francesco Nidito
Dipartimento di Informatica
Università di Pisa
nids@di.unipi.it

Michele Battelli
Google, Inc.
battelli@google.com

Stefano Basagni
Dept. of Electrical and Computer Engineering
Northeastern University
basagni@ece.neu.edu

***Abstract*— Heterogeneous wireless sensor networks are made up of different kinds of nodes. Some nodes, the *sensors*, are used as an interface to the physical environment. Other nodes act instead as servers, providing various services to the sensors. In this paper we define an architecture to enable the sensors to efficiently localize the services, and hence the servers. Our is a two-tier server architecture. The first tier is made up of the actual *servers*. The second tier is formed by nodes that are basically standard nodes (like the sensors). These nodes know the current position of the servers (they are called *server locators*). Sensors needing service query the server locators to find the corresponding service. The service locator sends a service position to the sensor. Finally, once got ahold of a server location, a sensor uses the service directly. Our server architecture provides load balancing (of queries to the servers) and is tolerant to server faults. Sensor nodes are endowed with caches to maintain the location of popular services. Experiments demonstrate the effectiveness of using caches at the sensor nodes.**

I. Introduction

Wireless Sensor Networks (WSNs) [1], [2] consist of a large number of low power, low cost and self-organizing wireless nodes forming a multi-hop (ad hoc) network [3]. The nodes are scattered (usually once and for all) in a given area without the support of any infrastructure. Therefore, they need to cooperate for executing common tasks, which usually consist in sensing environmental data and communicating to some collection points.

Heterogeneous WSNs are made up of various kinds of nodes. Some nodes are used as the interface to the physical environment (we will call them *sensors*). Other nodes acts instead as *servers*, providing services to the other nodes. For instance, in an outdoor intrusion detection application, the sensor are scattered randomly to provide tracking of possible intruders, while more powerful nodes provide the service or connection (e.g., through satellite links) to the end user.

In this paper, we define an architecture for enabling efficient discovery of services for the sensors that need them. Servers are organized into two tiers. The first tier comprises the actual servers. (as the satellite up-link enabled nodes mentioned before). Then there is an intermediate server tier (front-end) where some nodes (sensors, or more powerful nodes) are chosen for storing the current position of the servers. A sensor that needs to find a specific service (and hence a server), sends a message to the front-end of the system and gets back the position of the service. At this point the sensor knows the server position and is able to start using the needed service.

Our architecture implements strategies to provide both load balancing and fault-tolerance, as described in the rest of the paper. It also makes extensive use of a caching system to speed-up requests and to save energy. The localization/communication infrastructure used as the basis for our architecture is Q-NiGHT [4], [5], an improved instance of Geographic Hash Table, GHT [6].

The paper is organized as follows. Section II reviews GHT and Q-NiGHT, which are used as the building block of our architecture. Section III explains the architecture in detail. Section IV presents the experimental results. Finally, Section V concludes the paper.

II. GHT and Q-NiGHT

We briefly review here both GHT and Q-NiGHT, whose ideas and use inspired this work. The reader is referred to the cited paper for further references on this topic.

The idea on which our architecture rests is that of Geographic Hash Tables (GHT) [6]. More specifically, it uses a modified version of GHT called Q-NiGHT [4], [5]. The use of Q-NiGHT is motivated by the fact that it provides data QoS, even in networks where the nodes could be deployed non-uniformly. Specifically, the system provides the capability to specify the number of copies that are stored in the network for the data to be stored. Furthermore, being able to deal with non-uniform nodal distribution, Q-NiGHT enables load (storage) balancing since the data distribution fits the distribution of the nodes (or its approximation).

Q-NiGHT (as GHT) requires that the nodes are aware of their geographical position. This can be easily provided using a GPS system or other localization strategies [7]. For some applications, e.g., intrusion detection or some environmental monitoring, this knowledge is essential to provide meaningful data to the end-user.

Geographic Hash Tables are designed to enable efficient data storage and retrieval in the WSN itself. The basic operations are `put` and `get`. Data are represented by pairs $\langle key, value \rangle$, where the key identifies an item, and $value$ is the actual item to be stored. When a node performs a `put` to store data, the node uses a hash function on the item key. The hash function returns a pair of coordinates (x, y) within the

deployment area. At this time the pair $\langle key, value \rangle$ is routed (geographically [8]) toward (x, y) and it is stored at the nodes closer to that point. When a node performs a `get` operation to retrieve stored information with identifier key, the node uses the same hash function on key. The hash function returns the coordinate pair (x, y), and the node sends the request toward that point. As soon as the query is received at a node that stores the item sought for, that node replies with a packet containing the pair $\langle key, value \rangle$ to the requesting node.

Q-NiGHT improves on GHT in that it includes mechanisms for providing some QoS (in terms of fault tolerance), and better load balancing. In GHT the pair $\langle key, value \rangle$ is stored at the nodes on the perimeter around the point returned by the hash applied on key. We notice that GHT has no control on the number of copies of the stored data. Therefore, (*i*) GHT cannot guarantee fault-tolerance because data is replicated in a low number of copies, therefore, few faults can compromise it; (*ii*) GHT could store data in a large number of nodes (e.g., in high density networks), with corresponding unbalancing of the query load.

With Q-NiGHT data are stored at the Q closest nodes to the point returned by the GHT hash function (Q is therefore the number of copies of an item). With this simple extension, fault tolerance and load balancing are more easily achieved. Fault-tolerance comes with the fact that being stored at Q nodes, an item can survive $Q - 1$ faults. Load balancing is achieved via being able to control the replication, especially through using a new hash function. While GHT uses only uniform hashing to determine a point, Q-NiGHT uses a function that, using the knowledge about the distribution of the sensors (or an approximation) is able distribute data according to the distribution of the nodes. In particular, this function (REJECTIONHASH) is based on the rejection method for pseudo-random number generation [9], and it uses iterative hashing of the key, checking at each iteration the goodness of the returned value. The key is hashed over and over until it fits the wanted distribution (ever time being modified deterministically). When the hashed value fits the wanted distribution, the value returned as a valid coordinate pair for the item to be stored. Therefore, regions containing a larger number of nodes store a larger number of items, and regions with few nodes store fewer items.

As mentioned, further work on GHT and other methods for localizing services in sensor networks can be seen in [4]–[6].

III. SYSTEM ARCHITECTURE, OPERATIONS AND PROPERTIES

In this section we present our architecture for locating servers efficiently and in a fault-tolerant way.

We start by presenting the *actors* that play important roles in the architecture. These roles are both *structural* (given by the physical nature of the heterogeneous network) and *logical* (given the different usage of the same kind of nodes to perform different tasks).

We then present the operations of *service registration* and *look-up* that are provided by the system. The service registration procedure is performed by the servers to communicate their position to the nodes of the network. The look-up procedure is executed by the nodes that require the localization of a service.

A. Actors

There are three categories of nodes in our architecture. The *sensors*, the *servers*, and the *server locators*.

Sensors: The sensors are low power and low cost devices that are equipped with sensor to control their surrounding environment. They also sport a CPU for performing simple computations, and an embedded radio to communicate with each other.

Servers: The servers are special nodes that are capable to provide some service to the sensor nodes. These services range from storage (to keep the sensed data) to perform as gateways between the WSN and the users. Each server $server_i$ provides one or more services that are identified by a name, for instance $service_j$.

Server locators: The server locator nodes are nodes that know the servers, i.e., the services that the servers provide and the servers location in the network. These node can either be common sensors or more powerful nodes (although not necessarily as powerful as the servers).

B. Two-tiers servers architecture

The sensor are the clients of the proposed architecture. The servers are organized in two tiers. The *back-end* of the architecture is made up of the servers that are able to provide services to the clients. The servers are randomly deployed (as the clients) and need to be localized by the server locators. The server locators form the *front-end* of the architecture. The nodes requiring a service query the front-end to have the position of the the server, or servers, providing that service. Once obtained this information, the nodes communicate directly with the back-end servers.

C. Services localization

Two operations implement service localization: *Server registration* and *server discovery*. The first one is used by the servers to make the server locator aware of their location. The second one is used by the node that need a service for finding the corresponding server. These two operations use the two Q-NiGHT operations `put` and `get`. The first is for storing the position of a server and the second is used to retrieve it.

Servers registration: During the network set-up phase the server i determines its position, $position_i$, and registers it with the server locators nodes. To perform such operation each server hashes the name of each of its services and determines the corresponding point (x, y). At this time, for each one of the provided services, it performs a `put` of the pair $\langle service_j, position_i \rangle$ to the point (x, y) by using Q-NiGHT. The nodes that store the pair $\langle service_j, position_i \rangle$ become the server locator nodes for $service_j$.

Servers discovery: When a node needs $service_j$, it hashes the service name by using the Q-NiGHT hash function, finding the point (x, y). Then it performs a `get` operation of $service_j$ from point (x, y), i.e., it sends a request toward that point. One of the server locators replies with the position of the server (this operation is called a *look-up*). In addition to the basic Q-NiGHT `get` operation, at this time the node caches the position of the service/server for future use. It then sends the request for $service_j$ to the server. The use of caches improves the function of the whole look-up process in many ways. First of all, faster replies are provided to those other nodes interested in locating the same server whose look-up queries pass through the caching node. Moreover, caching enables cheaper look-ups because fewer hops can be enough to provide a reply, and lower energy consumption for the server locator nodes is required since they have to deal with a lower number of queries. For instance, cached positions increase information retrieval performance in applications such as intrusion detection. In this case, messages for a particular server are generated by nodes that are close to each other and to where the intrusion happens. Hence, spatial locality of caches is taken advantage of.

Fig. 1 depicts the interaction pattern between sensors, server locators and servers. The figure shows a situation in which a client first perform a service discovery sending a message to the server locator. The server locator replies with the server position. Finally, the node sends the necessary messages with the server. In particular, this picture refers to the case where

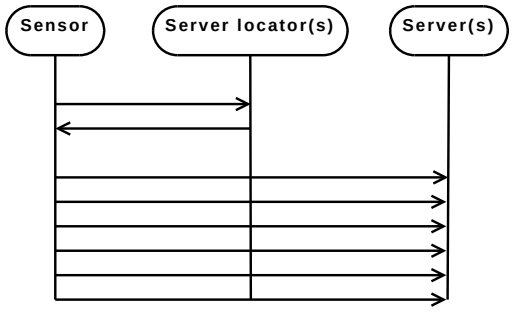

Fig. 1. Sensor, server locator and Server interaction pattern

the server is an "exit point" (like a gateway) for the network. Therefore, when the sensor has gained access to the server, it sends packet to it (the server does not send packets/acks back).

D. Load balancing and fault-tolerance

By using the Q-NiGHT mechanism described above, our architecture is able to balance the query load to the locator servers and to be tolerant to servers failures. The load balancing property is particularly useful for distributing multiple request of the same service to multiple servers that provide it. Fault-tolerance helps in removing from the list of services that provide a given service those servers that are no longer available (because of failures or network disconnections). The disappearance of a service/server can be signaled to a server locator by a sensor node that, trying to contact a server, realizes that it is no longer available. This feature of the architecture presents security issues, which we discuss at the end of this section.

Load balancing: Query load-balancing is provided via multiple server registrations. All the servers that provide a service have the same server locators. This happens because the servers share a common service name, say $service_j$. The server locators store all the coordinates that were provided for each service name. When a request arrives to a server locator for $service_j$, the node chooses one of the possible servers according to a given strategy (for instance, randomly, or in a a round-robin way, by keeping a pointer to the last server returned and incrementing it modulo the number of servers). This method also provides an easy way to increase the number of servers. When a new server (that provides $service_j$) enters the network, the server registers itself with the server locators and these return the server location as one of the possible servers for that service.

Fault-tolerance: Fault-tolerance to service outage is obtained as follows. The servers keep providing their position to the server locators periodically (for instance, each τ seconds).

In case of service failure, after τ seconds from the last update the server locators cancel the server position. In the worst case this system provides the cancellation of a server from server locators after 2τ seconds.

In order to make our protocol completely fault-tolerant we have to remove the cached server positions from the caches of the sensors that stored such information. To this aim, a server position is cached by a sensor for at most τ seconds, after which it is removed from the cache. In case the sensor needs the position of the server again it will have to query the server locators again. Finally, if a node queries a server whose entry was in its cache (i.e., τ seconds from its last query to a server locator have not passed yet) and the server is no longer available, the server's (ex) neighbors report an error to the sensor requesting the service. Upon receiving the error message the node removes the cache entry and performs a `get` for a new server for $service_j$, at the same time communicating to the server locators that the server is unavailable.

Security issues: The capability of sensors to invalidate a service location at a server locator, makes possible attacks in which the server locators server list are modified by malicious (or faulty) nodes.

To address this problem the network user (administrator) can choose between three solutions: (*i*) The sensors are not allowed to invalidate server locators and/or invalidation messages are dropped by server locators; (*ii*) The server locator verifies the invalidation querying the server itself to double check about the availability of that server, or (*iii*) the user provides a cryptography based system to verify the identity of the message sender and possibly its *trust-level*. All these solutions are equally able to provide a minimum level of trust to our architecture. The choice of one of them (or any combination of them) depends on the characteristics of the network (e.g., in terms of computational power and energy) as well as the

application and environmental characteristics (probably, for border monitoring or in the battlefield a level of trust much higher than wildlife monitoring is required).

IV. Experimental Results

We have performed experiments for measuring the effectiveness of our service localization architecture with respect to the energy cost of querying with and without caches, as well as the cost of the look-up operation.

In the simulation setting, we have considered 5000 wireless sensor networks where 5000 sensor nodes are scattered randomly and uniformly in a square area with a side long 1000m. Each node has communication range of 30m. Power consumption for transmission is set to 24mW and that for reception is set to 14.4mW, as in the EYES sensor prototypes [10]. The sensors that perform a look-up operation and then send a message to the server are uniformly chosen between the deployed sensors.

All the experiments are aimed at showing the effectiveness of the architecture in providing prompt and energy efficient response to sensor queries. In particular we show here that caching is particularly useful in providing a more balanced energy consumption, and therefore an overall better performance of the network. For this reason all the presented experiments are provided with and without nodal cache enabled. All tests are performed starting from the same seeds to generate the same scenarios with different architectural parameters. The results we show achieve a statistical confidence of 95%, with a precision within 5%.

Figures 2, 3 and 4 depict the cost for a sensor to contact server locators and servers. In Fig. 2 the cost is defined as the energy spent by a node to send a packet to the server locators, to get the server location back and then to perform one communication to the server. In other words we compute the total energy to deliver/receive three packets. In Fig. 3 the cost is defined only as the cost to send a message from a sensor to the server locators and to get the server location back (that is, the energy to send two packets). This provides us with a more detailed idea of how much it costs to a sensor to use the intermediate tier provided by the sensor locators. Fig. 4 shows the cumulative cost of the look-up operation, to have an idea of difference in the growth of the energy cost. As mentioned, each set of experiments is performed with and without the cache mechanism enables. The network is observed for a time long τ to take into account the maximum usage of the caches before their refresh. In our experiments $\tau = 30$ minutes i.e., the time needed to perform 1000 queries.

Fig. 2 depicts the cost for each single query in the network with and without the caching enabled. This cost, expressed in Jules, is defined as the sum of the energy spent at each node for propagating the query. This cost is computed considering both the cost for transmission and reception We observed that when caching is enabled the cost of the single query decreases with increasing number of queries because the caching mechanism becomes more and more effective (more and more nodes have the location in cache).

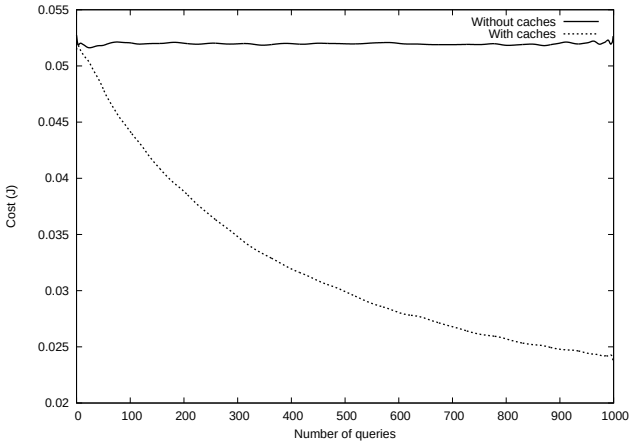

Fig. 2. Cost for server look-up and for contacting the servers

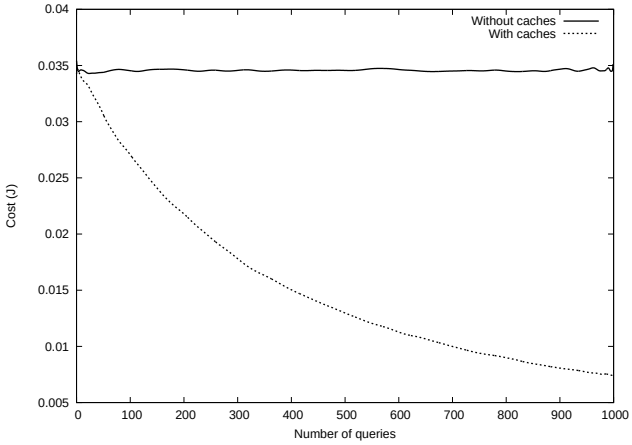

Fig. 3. Cost for servers look-up operations.

Fig. 3 depicts only the cost of the look-up operation for each single query in the network with and without the caching enabled on sensors. This cost, as the previous one, is defined as the sum of the energy spent at each node for propagating the query. This case, is used to have a better evaluation of the look-up procedure, that is cached optimized, with respect to the server interaction, that in our scenario does not use caches to optimize the sensor-server. Some particular applications can use caching also between sensors and servers but we chose this situation (the worst case) in which the interaction with the server is not cached to have a more clear vision of the look-up costs and benefits.

Fig. 4 depicts the cumulative cost of the look-ups only (sensor-server locators communication and back) without considering the cost for server interaction. This cost (in Joules) is computed as follows: The cost of the qth look-up is given by its cost and summed to the cost of all the previous $q-1$ look-ups. In this case we observe that the cost to reach the server locator nodes decreases when the number of the queries increase, as expected.

Fig. 5 presents the (normalized) residual energy level of the server locators with and without using caches. The residual

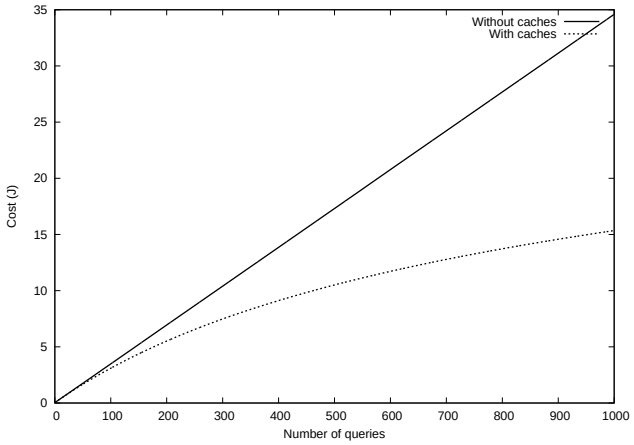

Fig. 4. Cumulative cost for servers look-up operations.

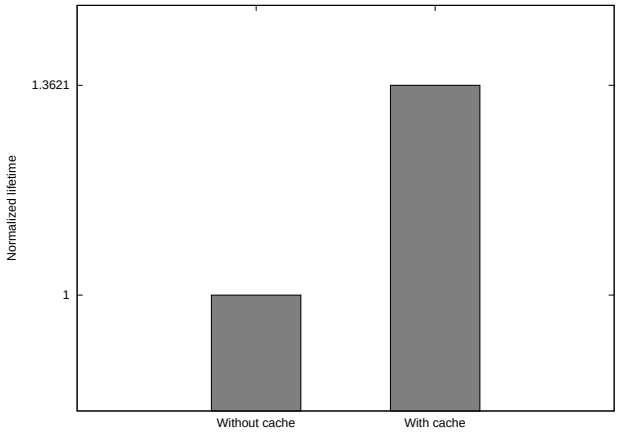

Fig. 5. server locators residual energy level at the end before the server locators refresh step(normalized).

energy level of the server locator is computed reading the energy level of the server locators before the first refresh of the service location by the servers, that restore Q copies of the location, also in the case in which some server locators run out of energy. The figure shows a high influence (36%) of the caches in the energy spent by the server locators in their function.

V. Conclusions and Future Work

In this paper we presented an architecture to provide server localization in heterogeneous wireless sensor networks. The proposed architecture enables servers to register the services that they provide, with their location, in some algorithmically elected nodes (the server locators) that sensor nodes can find easily. A sensor needing a service is able, using the name of the service, to locate a group of nodes (and their position) that know the actual current position of the service.

They can therefore query them to obtain the position of the server providing the service. At this point the node is able to query the server for its service. The presented architecture provides this service in a load-balanced and fault-tolerant way. A caching system enables the nodes that lay in the same region to assist in providing service location while relieving the service locators of providing the location. The effectiveness of using caches is demonstrated by experiments.

A more detailed performance evaluation remains to be performed where the cost metrics actually take into account physical and MAC layer characteristics of WSNs. Moreover, we want to investigate the performances of our system in the case in which other routing protocols are used (e.g., [11], [12]). Furthermore, we want to investigate more in depth the security aspects of the system, as well as the possibility to use other load balancing and fault-tolerant strategies. Finally, the obtained results are encouraging and open up possibilities for further studies and for the application of this method to problems such as data storage and replication.

References

[1] I. F. Akyildiz, W. Su, Y. Sanakarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, March 2002.

[2] F. Zhao and L. Guibas, *Wireless Sensor Networks, An Information Processing Approach*, ser. Networking. San Francisco, CA: Morgan Kaufmann, January 2004.

[3] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, Eds., *Mobile Ad Hoc Networking*. Piscataway, NJ and New York, NY: IEEE Press and John Wiley & Sons, Inc., April 2004.

[4] M. Albano, S. Chessa, F. Nidito, and S. Pelagatti, "Q-NiGHT: Adding QoS to Data Centric Storage in Non-Uniform Sensor Networks," Dipartimento di Informatica, Università di Pisa, Tech. Rep. TR-06-16, 2006.

[5] ——, "Q-NiGHT: Adding QoS to Data Centric Storage in Non-Uniform Sensor Networks," in *Proc. of the 8th International Conference on Mobile Data Management (MDM'07)*, Mannheim, Germany, May 2007, pp. 166–173.

[6] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensornets with GHT, a geographic hash table," *Mobile Networks and Applications (MONET)*, vol. 8, no. 4, pp. 427–442, 2003.

[7] M. Battelli and S. Basagni, "Localization for wireless sensor networks: Protocols and perspectives," in *Proceedings of IEEE CCECE 2007*, Vancouver, Canada, April 22–26 2007.

[8] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proc. of the 6th International Conference on Mobile Computing and Networking (MobiCom 2000)*, Boston, MA, USA, August 2000, pp. 243–254.

[9] J. V. Neumann, "Various techniques used in connection with random digits," in *John von Neumann, Collected Works*, A. H. Taub, Ed. Oxford: Pergamon Press, Oxford, 1951, vol. 5, pp. 768–770.

[10] P. J. M. Havinga, S. Etalle, H. Karl, C. Petrioli, H. K. M. Zorzi, and T. Lentsch, "Eyes–energy efficient sensor networks," in *Proceedings of PWC 2003*, Venice, Italy, September 2003, pp. 198–201.

[11] M. Zorzi and R. R. Rao, "Geographic random forwarding (GeRaF) for ad hoc and sensor networks: Multihop performance," *IEEE Transactions on Mobile Computing*, vol. 2, no. 4, pp. 337–348, October-December 2003.

[12] P. Casari, M. Nati, C. Petrioli, and M. Zorzi, "Efficient non-planar routing around dead ends in sparse topologies using random forwarding," in *Proceedings of the IEEE International Conference on Communications, ICC 2007*, Glasgow, Scotland, June 24–28 2007.