Coordinated data aggregation in wireless sensor networks using the omega failure detector *

Mikel Larrea[1], Cristian Martín[1], and José Javier Astrain[2]

[1] Computer Architecture and Technology Department
The University of the Basque Country
20018 San Sebastián, Spain
{mikel.larrea, acbmahec}@ehu.es
[2] Departamento de Matemática e Informática
Universidad Pública de Navarra
31006 Pamplona, Spain
josej.astrain@unavarra.es

Abstract. We present an algorithm implementing the failure detector class omega (Ω) in the crash-recovery model to coordinate data aggregation in wireless sensor networks. In order to minimize the energy consumption when collecting sensor data of a wireless sensor network, we divide the network in regions. The algorithm ensures the agreement on a common aggregator by all sensors of a region regarding sensor reachability and power saving criteria. Data aggregation is performed by the sensor aggregator selected by the algorithm.

Keywords: data aggregation, wireless sensor networks, agreement, failure detectors.

1 Introduction

The emergence of wireless sensor networks is due to the demand of reliable data collection by applications. One of the issues of a wireless sensor network is to reduce at maximum the human intervention (self-organizing and self-maintaining), since only deployment and maintenance tasks are performed occasionally. Applications can act autonomously over the sensor network configuring sensors remotely, and recovering the information collected by sensors periodically or on demand.

Two important factors to be considered on wireless sensor networks are: i) the overall deployment cost, which can be reduced using low cost sensors nodes; and ii) the operation lifetime limited by the energy consumption, which can be reduced minimizing the operation time and enabling sensor hibernation. In this way, wireless networked sensing applications must ensure reliable sensor

* Research partially supported by the Spanish Research Council MCyT (TIN2004-07474-C02-02 and TIC2003-09420-C02), the Basque Government (S-PE05IK06) and the Gipuzkoa Provincial Council (OF-204/2005).

data collection and aggregation, while satisfying the low-cost and low-energy operating constraints of such applications. In this work we focus on this second topic.

The attempt to minimize the energy consumption leaves to minimize the amount of data transmitted by using data aggregation. Some works focus on the implementation of efficient aggregation timing control protocols as in [5]. In [14], three different data aggregations schemes: in-network, grid-based and hybrid data aggregations are considered in order to increase the throughput, to decrease congestion and to save energy. Other works manage data aggregation with the aid of a consensus algorithm as [8] does. Sensor data can be collected and aggregated by certain nodes of the sensor network called *aggregators* [13]. The selection of the aggregator node is analyzed in [11] where a hierarchical energy-efficient aggregation protocol is presented.

Wireless sensor networks can be exposed to several sources of problems such as measuring, communication, crash and/or power supply errors. Device redundancy allow to obtain information redundancy, and then to ensure a certain level of fault tolerance. We consider a wireless sensor network with a certain clustering degree where each cluster focus on specific areas or events and collects information for a collecting base station, following a distributed sensing scheme. It also enables a best dynamic reaction to scenario events, since some sensors can be readapted to new demands.

The concept of unreliable failure detector introduced by Chandra and Toueg in [4] can be used to solve several problems in asynchronous distributed systems, in particular the Consensus problem [12]. In this paper we use the failure detector omega [3], denoted Ω, to select aggregator nodes in a wireless sensor network. Ω provides eventual agreement on a common leader among all non-faulty processes in a system and is the weakest unreliable failure detector for solving consensus. Several algorithms implementing Ω have been proposed in the literature. In [9] Larrea et al. propose an algorithm which requires all links (incoming and outgoing) to be eventually timely. In [1] only some unknown correct node must have all its links eventually timely. Furthermore, Aguilera et al. consider the outgoing links from some unknown correct node to be eventually timely in [2]. The rest of links, called *lossy asynchronous*, can lose messages and/or arbitrarily delay them.

All the above mentioned implementations of Ω assume membership knowledge. In wireless sensor networks, due to their dynamic nature, it could be interesting to relax this requirement, i.e., not to restrict the membership to an *a priori* known set of sensors. In this sense, Jiménez et al. propose in [6] the first Ω algorithm with unknown (but finite) membership for a crash failure model. In [10] we adapt this algorithm to the crash-recovery failure model. Moreover, both algorithms work in systems in which not every pair of processes is connected by a direct communication link, i.e., systems with partial connectivity.

In this paper, we investigate the use of Ω in wireless sensor networks to coordinate data aggregation. More precisely, we propose an adaptation of the algorithm of [10] for a system divided in regions, where every sensor can reach

directly the rest of sensors located in its region. Due to changes in the set of reachable sensors or to power saving criteria, the aggregator sensor of each region can change over time. Ω ensures the agreement on a common aggregator by all sensors of a region.

The rest of the paper is organized as follows. In Section 2, we describe the system model considered in the paper. In Section 3, we present the algorithm implementing Ω. Finally, conclusion and future work end the paper.

2 System model

We consider a wireless sensor network where we try to collect sensor data minimizing the energy consumption. For such purpose, we establish some time slots used by sensors to provide the sensed information. Sensors hibernate the rest of the time. This assumption implies a programmed switch on/off of sensors with the aid of a clock. We assume that hibernation periods are larger than active ones (see Figure 1). Each sensor has its own clock, which can be synchronized periodically on demand by the system administrator. As clock drift is common, we assume a maximum clock skew between any pair of sensors, denoted ϵ, that depends on the type of clock used by sensors, and the time interval considered. Sensors must take into account: i) clock skew, ii) the time needed to collect sensor data, and iii) the time needed to ensure the correct operation of the Ω algorithm. Figure 1 illustrates the system operation timeline.

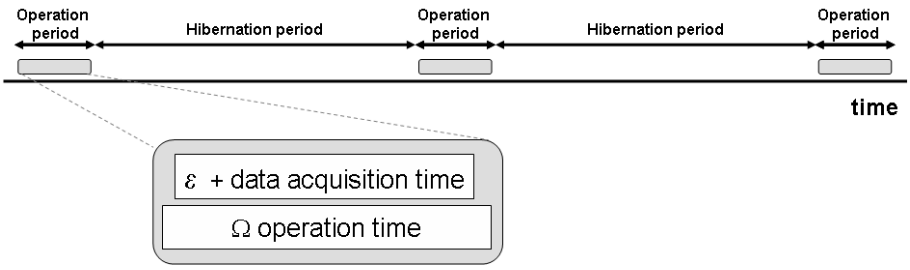

Fig. 1. System operation timeline.

Sensor data collection can follow an address-centric model, or a data-centric model [7]. We consider a data-centric model to perform sensor data aggregation attempting to power saving criteria. Let be a wireless sensor network divided in different regions where each sensor knows *a priori* its operation region and deals to transmit its sensed data to a sensor aggregator in charge of the collection of all the sensed data from a specific region. The aggregator can be anyone of the sensors of a region, and it can be selected dynamically according to different criteria. Each sensor has an identifier of its region operation that is radiated in its messages. This identifier allows sensors the rejection of incoming messages

from other regions. The region operation of a sensor can be changed on demand, as well as the region definition.

We propose an implementation of Ω to choose a common aggregator inside each region according to the reliability and battery availability of sensors. A sensor is a candidate to be elected as aggregator if it is a reliable sensor without errors during each operation period, and it has a good battery availability. Ω ensures that a unique sensor among all the candidates is elected as aggregator. The aggregator of each region collects all the data sensed on its coverage area.

Our Ω implementation, based on [6, 10], considers a system S composed of a finite set Π of $n > 1$ sensors $\Pi = \{s_{id_1}, s_{id_2}, \ldots, s_{id_n}\}$, that communicate only by sending and receiving messages. The sensor identifiers are totally ordered, but need not be consecutive. All the sensors of a region can communicate directly, existing an unknown bound Δ on message delay.

Sensors can only fail by crashing. Crashes are not permanent, i.e., crashed sensors can recover. We consider as a particular type of crash failure the depletion of the battery of a sensor, being the hypothetical replacement of the battery its associated recovery. Sensor hibernation is not considered a failure, since it is a scheduled task. According to the previous, and during the lifetime of its battery, a sensor can be:

(1) *Eventually up.* It is a sensor that, after crashing and recovering a finite number of times, remains up until the end of its battery. Sensors that never crash belong to this class.
(2) *Eventually down.* It is a sensor that, after crashing and recovering a finite number of times, remains down until the end of its battery. Sensors that never start their execution belong to this class.
(3) *Unstable.* It is a sensor that crashes and recovers continuously until the end of its battery.

By definition, sensors in (1) are *correct*, and sensors in (2) and (3) are *incorrect*. We assume that the number of correct sensors in the system is at least one. We also assume that every sensor has access to stable storage to keep the value of some variables. As we will see, correct sensors will be the candidates to become the leader, i.e., the aggregator.

The Ω failure detector

Chandra, Hadzilacos and Toueg defined in [3] the Ω failure detector. The output of the failure detector module of Ω at a process p is a single process q, that p currently considers to be correct (we say that p *trusts* q). The failure detector Ω satisfies the following property:

Property 1. There is a time after which all the correct processes always trust the same correct process.

Note that the output of the failure detector module of Ω at a process p may change through the time, i.e., p may trust different processes at different times.

Furthermore, at any given time t, two processes p and q may trust different processes.

Observe that this definition of Ω considers that eventually the common leader holds *forever*. In our system model, due to the limited duration of the battery of sensors, this definition is not adequate for our purpose. We redefine Ω in Property 2 considering that eventually the common leader holds until the end of its battery.

Property 2. There is a time after which all the correct sensors trust the same correct sensor *aggregator* until the end of its battery.

3 The algorithm

In this section we present an algorithm that implements in system S the definition of Ω described in Property 2.

Figure 2 presents the algorithm in detail. The sensor chosen as aggregator by a sensor p, i.e., trusted by p, is held in a variable $leader_p$. The algorithm uses both stable and volatile storage. Variables include a (private) incarnation number, initialized to 0, which is incremented during initialization and every time it recovers from a crash. Variables $incarnation_p$, $leader_p$, $incarnation_{leader}$ and $Timeout_p$ are persistently stored while variable $scheduled\_wakeup_p$ remains in volatile storage. Besides this, every sensor has a local timer used to detect the potential crash of the aggregator sensor. During hibernation periods the variable $scheduled\_wakeup_p$ keeps its value, while sensor failure or battery depletion causes the loose of its value, i.e., $scheduled\_wakeup_p = FALSE$. Constant $\Delta_{TIMEOUT}$ determines the growth of the timeout in order to reach agreement. The higher this value is, the faster agreement on a common aggregator occurs. Constant $\Delta_{DATA\_ACQUISITION}$ represents the maximum time spent by the aggregator during the collection of the data provided by sensors. Finally, all sensors are pre-programmed to wake-up periodically, e.g., every $\Delta_{ACTIVATION}$ time units.

Let us denote by c_{min} the correct sensor in S with the smallest identifier among those that have the minimum incarnation number $incarnation_{min}$. We will show that with this algorithm there is a time after which every sensor $p \in correct$ permanently has $leader_p = c_{min}$, and hence satisfies Property 2. Eventually only sensor c_{min} broadcasts new leader messages (I-AM-THE-LEADER, c_{min}, $incarnation_{min}$) twice per operation period (taking into account the clock drift), that reach the rest of correct sensors. The message is broadcasted twice in order to minimize the effect of the clock drift, since a sensor can wake-up with a maximum delay of ϵ from the first broadcast of the message. Lines 17 and 19 correspond to these broadcasts.

We assume that for any sensor to start sending (I-AM-THE-LEADER, -, -) messages (Lines 17 or 19), it has necessarily incremented its incarnation number by one in stable storage during initialization (Line 7) if the wake-up of the sensor from the hibernation state was not scheduled. The replacement of the depleted

Every process p executes the following:

procedure $GoToHibernation()$
(1) write ($incarnation_p$, $leader_p$, $incarnation_{leader}$, $Timeout_p$) in stable storage
(2) $scheduled\_wakeup_p \leftarrow TRUE$
(3) $hibernate()$
end procedure

Initialization:
(4) read ($incarnation_p$) from stable storage
(5) **if** $[scheduled\_wakeup_p = FALSE]$ **then**
(6) $incarnation_p \leftarrow incarnation_p + 1$
(7) write $incarnation_p$ in stable storage
(8) **end if**
(9) $scheduled\_wakeup_p \leftarrow FALSE$
(10) read ($leader_p$, $incarnation_{leader}$, $Timeout_p$) from stable storage
(11) **if** $leader_p = p$ **then**
(12) **start tasks** 1 and 2
(13) **else**
(14) reset $timer_p$ to $Timeout_p + \epsilon$
(15) **start tasks** 2 and 3
(16) **end if**

Task 1:
(17) broadcast (I-AM-THE-LEADER, p, $incarnation_p$)
(18) receive data from sensors during ϵ time
(19) broadcast (I-AM-THE-LEADER, p, $incarnation_p$)
(20) receive data from sensors during $\Delta_{DATA\_ACQUISITION}$ time
(21) $GoToHibernation()$

Task 2:
(22) **upon reception of** message (I-AM-THE-LEADER, q, $incarnation_q$) such that $[incarnation_q < incarnation_{leader}]$ or $[(incarnation_q = incarnation_{leader})$ and $(q \leq leader_p)]$ **do**
(23) $leader_p \leftarrow q$
(24) $incarnation_{leader} \leftarrow incarnation_q$
(25) send data to $leader_p$
(26) $GoToHibernation()$

Task 3:
(27) **upon expiration of** $timer_p$ **do**
(28) $leader_p \leftarrow p$
(29) $incarnation_{leader} \leftarrow incarnation_p$
(30) $Timeout_p \leftarrow Timeout_p + \Delta_{TIMEOUT}$
(31) $GoToHibernation()$

Fig. 2. Implementation of Ω in a wireless sensor network.

battery of a sensor also implies the increment of its incarnation number by one, since the $scheduled\_wakeup_p$ variable is placed in a volatile storage and the replacement of the battery resets the sensor (takes value $FALSE$ when restarting).

The system considers as a crash failure of the sensor the full depletion of its battery. This failure becomes a crash-recovery failure if a replacement of this battery occurs. Some sensors can be easily reachable making possible the replacement of the batteries, but other sensors can be unreachable.

Algorithm starts with *Initialization* where all the values of the variables are properly recovered. After that, if the sensor considers itself as the current aggregator, the algorithm starts tasks 1 and 2. On the other hand, if the sensor does not consider itself as the current aggregator, the algorithm resets the local timer and starts tasks 2 and 3. Task 1 is devoted to announce the aggregator to the rest of sensors and to collect all sensor data. Task 2, which applies to all sensors, is devoted to both send the sensor data to the aggregator, and to update the aggregator if required. Finally, Task 3 is devoted to propose the sensor itself as aggregator when the current aggregator announcement is not received before the expiration of the timer, and also increments the timeout $\Delta_{TIMEOUT}$ time units. All the tasks finish calling the *GoToHibernation()* procedure, which starts the hibernation period. Each sensor will remain in this state until the next scheduled wake-up.

Sensors execute by taking steps. We assume the existence of a lower bound σ on the number of steps per unit of time taken by any correct sensor. For simplicity, we assume that each line of the algorithm represents one step. Sensors have clocks that accurately measure intervals of time but are not necessarily synchronized, although the maximum clock skew is known.

For the rest of the section we will assume that any time instant t is larger than a time $t_{base} > t'_{base}$. Time t'_{base} is a time instant that occurs after every eventually down sensor has definitely crashed, every eventually up sensor has definitely recovered, and every unstable sensor has an incarnation number bigger than $incarnation_{min}$. Time t_{base} occurs after every message sent before $t'_{base}+3\sigma$ have already been delivered.

In order to prove the correctness of the algorithm, we formulate and prove the following lemmas and theorem. Lemma 1 proves that sensor c_{min} becomes an aggregator and notifies this fact to the rest of sensors. Lemma 2 proves that the rest of sensors do not declare themselves as aggregators. Lemma 3 proves that there is a time after which every correct sensor receives new (I-AM-THE-LEADER, c_{min}, $incarnation_{min}$) messages from the aggregator c_{min}. Finally, Theorem 1 proves that the algorithm of Figure 2 implements Ω in system S following the definition described in Property 2.

Lemma 1. *There is a time after which sensor c_{min} permanently verifies that $leader_{c_{min}} = c_{min}$ and broadcasts the* (I-AM-THE-LEADER, c_{min}, $incarnation_{min}$) *message twice during its operation period.*

Proof. Note that after time t_{base}, sensor c_{min} will never receive a message (I-AM-THE-LEADER, q, $incarnation_q$) with $incarnation_q < incarnation_{min}$, or

with $incarnation_q = incarnation_{min}$ from a sensor with identifier $q < c_{min}$. Therefore, after time t_{base} sensor c_{min} will never execute Lines 23-26 of the algorithm. Hence once $leader_{c_{min}} = c_{min}$ it will remain so until the depletion of its battery. To show that this eventually happens, note that if $leader_{c_{min}} \neq c_{min}$ at time $t > t_{base}$, then c_{min} either has $timer_{c_{min}}$ active, or will reset it during its initialization (Line 14). In both cases, $timer_{c_{min}}$ will eventually expire (Line 27), setting $leader_{c_{min}} = c_{min}$ (Line 28). After that, Lines 14-15 will never be executed, since $leader_{c_{min}} = c_{min}$ holds permanently. Finally, from Task 1, once $leader_{c_{min}} = c_{min}$, process c_{min} will permanently broadcast twice the (I-AM-THE-LEADER, c_{min}, $incarnation_{min}$) message during its operation period (Lines 17 and 19).

Lemma 2. *There is a time after which, every sensor $p \in correct$, $p \neq c_{min}$, permanently has either (1) $incarnation_{leader} > incarnation_{min}$, or (2) $leader_p \geq c_{min}$ and $incarnation_{leader} = incarnation_{min}$.*

Proof. Note that, after t_{base}, once $[incarnation_{leader} > incarnation_{min}]$ or $[(incarnation_{leader} = incarnation_{min})$ and $(leader_p \geq c_{min})]$ is satisfied, it will remain so until the depletion of the battery (either the sensor or the aggregator), since no (I-AM-THE-LEADER, q, $incarnation_q$) message with $incarnation_q < incarnation_{min}$, or with $incarnation_q = incarnation_{min}$ from a sensor with identifier $q < c_{min}$ will be received. Then, if $incarnation_{leader} < incarnation_{min}$, or $incarnation_{leader} = incarnation_{min}$ and $leader_p < c_{min}$ at time $t > t_{base}$ in both cases it is satisfied that (1) $incarnation_p > incarnation_{min}$, or (2) $p > c_{min}$ and $incarnation_p = incarnation_{min}$. Then, $timer_p$ must be active at that time, and will eventually expire (Line 27), setting by Lines 28-29 either (1) $incarnation_{leader} = incarnation_p > incarnation_{min}$, or (2) $incarnation_{leader} = incarnation_p = incarnation_{min}$ and $leader_p = p > c_{min}$.

Lemma 3. *There is a time after which every sensor $p \in correct$, $p \neq c_{min}$, permanently receives new* (I-AM-THE-LEADER, c_{min}, $incarnation_{min}$) *messages with intervals of at most $(\Delta_{ACTIVATION} + \Delta + 2\epsilon + 8\sigma)$ time between consecutive messages.*

Proof. From Lemma 1, there is a time after which c_{min} broadcasts two new (I-AM-THE-LEADER, c_{min}, $incarnation_{min}$) messages every operation period. For simplicity, let us assume that p received the last (I-AM-THE-LEADER, c_{min}, $incarnation_{min}$) message from c_{min} at the beginning of the previous operation period. In the worst case, and due to clock drift, c_{min} will wake-up after almost $\Delta_{ACTIVATION} + \epsilon$ time. If c_{min} wakes-up before p and the first (I-AM-THE-LEADER, c_{min}, $incarnation_{min}$) message it broadcasts after 7σ (Line 17) is not received by p, it takes to c_{min} $\epsilon + \sigma$ additional time to broadcast its second (I-AM-THE-LEADER, c_{min}, $incarnation_{min}$) message. This message takes at most Δ time to reach sensor p. Henceforth, the maximum time between two consecutive messages is the addition of these values $\Delta_{ACTIVATION} + \Delta + 2\epsilon + 8\sigma$.

Theorem 1. *There is a time after which every sensor $p \in correct$ has $leader_p = c_{min}$, i.e., p trusts c_{min}, until the end of either p's or c_{min}'s battery. Hence, the algorithm of Figure 2 implements Ω (Property 2) in system S.*

Proof. Lemma 1 shows the claim for $p = c_{min}$. For $p \neq c_{min}$, from Lemma 2 there is a time after which p permanently (until the end of the battery) has either (1) $incarnation_{leader} > incarnation_{min}$, or (2) $incarnation_{leader} = incarnation_{min}$ and $leader_p \geq c_{min}$. From Lemma 3, whenever $leader_p \neq c_{min}$ after this time, $leader_p$ changes to c_{min} in at most $(\Delta_{ACTIVATION} + \Delta + 2\epsilon + 8\sigma)$ time ($+2\sigma$ for the assignation of the new leader). Furthermore, once $leader_p = c_{min}$, it only changes (to p) by executing Lines 27-28, since the conditions in Line 22 prevent $leader_p$ from changing in Line 23. Finally, $leader_p$ changes from c_{min} to p a finite number of times, since each time this happens $Timeout_p$ is incremented by $\Delta_{TIMEOUT}$ time units. By contradiction, assuming this happens an infinite number of times, $Timeout_p$ eventually grows to the point in which $timer_p$ never expires, because new (I-AM-THE-LEADER, $c_{min}, incarnation_{min}$) messages are received often enough before the expiration of $timer_p$. Hence, eventually $leader_p = c_{min}$ permanently, and thus the algorithm of Figure 2 implements Ω (Property 2) in system S.

4 Conclusion

In this paper, we have presented an algorithm that implements the failure detector class omega (Ω) in the crash-recovery model for wireless sensor networks. This algorithm allows the selection of a common aggregator sensor in charge of the data collection of its influence region.

5 Future work

The algorithm presented in this paper is focused on the acquisition of sensor data in a given region of a local wireless sensor network. Applying the algorithm to the aggregators of the different regions we can collect all the data supplied by the sensors of a wide area sensor network. The selection process can be performed in two steps, the first one to select the aggregator in a region, and the second one to select the super-aggregator among all the local aggregators.

References

1. M. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Stable leader election. In *Proceedings of the 15th International Symposium on DIstributed Computing (DISC'2001)*, pages 108–122, Lisbon, Portugal, October 2001. LNCS 2180, Springer-Verlag.
2. M. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. On implementing Ω with weak reliability and synchrony assumptions. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC'2003)*, pages 306–314, Boston, Massachusetts, July 2003.
3. T.D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

4. T.D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.
5. Fei Hu, Xiaojun Cao, and Carter May. Optimized scheduling for data aggregation in wireless sensor networks. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)*, volume 2, pages 557–561, Las Vegas, Nevada (USA), April 2005. IEEE Computer Society Press.
6. E. Jiménez, S. Arévalo, and A. Fernández. Implementing the Ω Failure Detector with Unknown Membership and Weak Synchrony. Technical Report RoSaC-2005-2, Universidad Rey Juan Carlos, Spain, January 2005.
7. B. Krishnamachari, D. Estrin, and S. Wicker. Modelling data centric routing in wireless sensor networks. Technical Report CENG 02-14, USC Computer Engineering, 2002.
8. M. Kumar, L. Schwiebert, and M. Brockmeyer. Efficient data aggregation middleware for wireless sensor networks. In *Proceedings of the First International Conference on Mobile, Ad-Hoc, and Sensor Systems*, pages 579–581, Ft. Lauderdale, Florida (USA), October 2004. IEEE Computer Society Press.
9. M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'2000)*, pages 52–59, Nurenberg, Germany, October 2000.
10. M. Larrea and C. Martín. Implementing the Ω failure detector in the crash-recovery model with partial connectivity and/or synchrony. In *Proceedings of the 2nd International Workshop on High Availavility of Distributed Systems (HADIS'2006)*, Krakow, Poland, Sep 2006. To appear. Technical Report available at http://www.sc.ehu.es/acwlaalm/.
11. Y.P. Chen A.L. Liestman and J. Liu. Energy-efficient data aggregation hierarchy for wireless sensor networks. In *Proceedings of the 2nd International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QShine'05)*, page 7, Lake Buena Vista, Florida (USA), August 2005. IEEE Computer Society Press.
12. M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.
13. B. Przydatek, Dawn Xiaodong Song, and A. Perrig. Sia: secure information aggregation in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys 2003*, pages 255–265, Los Angeles, California (USA), November 2003. ACM.
14. Karthikeyan Vaidyanathan, Sayantan Sur, Sundeep Narravula, and Prasun Sinha. Data aggregation techinques in sensor networks. Technical Report 11/04-TR60, OSU-CISRC, 2004.