RAPPORT DE RECHERCHE

L R I

SELF-STABILIZING COUNTING IN MOBILE SENSOR NETWORKS

BEAUQUIER J / CLEMENT J / MESSIKA S / ROSAZ L / ROZOY B

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud – LRI

03/2007

Rapport de Recherche N° 1470

CNRS – Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 490
91405 ORSAY Cedex (France)

Self-Stabilizing Counting in Mobile Sensor Networks

Joffroy Beauquier, Julien Clément, Stephane Messika, Laurent Rosaz, Brigitte Rozoy

LRI, CNRS & Univ. Paris Sud,
91405 Orsay Cedex, France
Contact person: J. Clément, e-mail: clement@lri.fr,number of pages:12

Abstract. Distributed computing must adapt its techniques to networks of mobile agents. Indeed, we are in front of new problems like the small size of memory and the lack of computational power. In this paper, we extend the results of Angluin et al (see [1–3]) by finding self-stabilizing algorithms to count the number of agents in the network. We focus on two different models of communication, with a fixed antenna or with pairwise interactions. In both models we decide if there exist algorithms (probabilistic, deterministic, with k-fair adversary) to solve the self-stabilizing counting problem.

Key-words: distributed algorithms, fault-tolerance, self-stabilization, sensor networks, mobile networks
Eligible for Brief Announcement

1 Introduction

Habitat and environmental monitoring represents a class of sensor network applications with enormous potential benefits both for scientific communities and society as a whole. The intimate connection with its immediate physical environment allows each sensor to provide localized measurements and detailed information that is hard to obtain through traditional instrumentation. Many environmental projects use sensor networks.
The SIVAM project in Amazonia is related to meteorological predictions, sensors are placed in glacial areas for measuring the impact of the climate evolution, (see [7]), use of sensors is considered in Mars exploration (see [8]) or for detecting the effect of the wind or of an earthquake on a building (see [9]).
A sensor network has been deployed on Great Duck Island (see [6]) for studying the behavior of Leachs Storm Petrels. Seabird colonies are notorious for the sensibility to human disturbance and sensor networks represent a significant advance over traditional methods of monitoring. In [2], Angluin et al. introduced the model of population protocols in connection with distributed algorithms for mobile sensor networks. A sensor is a package containing power supply, processor, memory and wireless communication capacity. Some physical constraints involve limitations of computing or storage capacity and communication. In particular two sensors have to be close enough to be able to communicate. A particular static entity, the base station (or antenna), is provided with more computing resources.
A computation starts with the base station sending a global signal reaching all sensors. When it receives the signal the sensor reads an input value (the value of what it is supposed to observe) and executes its code. The code defines what happens when two close sensors communicate and how they communicate with the base station. An important assumption made in this model is that the interactions between the sensors themselves and between the sensors and the base station are not controlled. Also, a hypothesis of fairness states that in an infinite computation the numbers of interactions between two given sensors or between a particular sensor and the base station are infinite. Eventually the result of the computation is stored at the base station and does not change any more.
This model takes into account the inherent power limitation of the real sensors and also the fact that they can be attached to unpredictably moving supports. For being still more realistic the model should consider the possibility for the sensors to endure failures. Temperature variations, rain, frost, storm, etc. have as a consequence, in the real world, that some sensors are crashed and that some others are still operating, but with corrupted data.
Most of population protocols do not consider the possibility of failures. The aim of this paper is to perform computation in mobile sensor networks subject to some type of failures. The framework of self-stabilization is particularly well adapted for dealing with such conditions. A self-stabilizing system, after some memory corruptions hit part of all processors, regains its consistency by itself, meaning that first,

(convergence) it reaches a correct global configuration in a finite number of steps and second, (correction) from then its behavior is correct until the next corruption. It is important to note that in this model, the code is supposed to be stored in a ROM and then cannot be corrupted. Traditionally self-stabilization assumes that failures are not too frequent (for giving enough time to the system for recovery) and thus the effect of a single global failure is considered. That is equivalent to consider that the system may be started in any possible global configuration. Note that the issue of combining population protocols with self stabilization has been addressed for ring networks in [1] and in a different framework in [4].
In the present work we make the assumption that, if the input variables can be corrupted, as any other variable, first they do not change during the time of the computation and second they are regularly read by the sensor. Then eventually a sensor deals with its correct input values.
In this paper we consider the very basic problem of computing the number of (not crashed) sensors in the system, all sensors being identical (same code, no identifiers), when their variables are arbitrary initialized (but the input value of each sensor is 1). This problem is fundamental, first because the ability of counting makes easier the solution of other problems (many distributed algorithms assume that the size of the network is known in advance) and second because if counting is feasible, sum, difference and test to 0 are too. In practice, one might want to count specific sensors, for example those carried by sick petrels.
We present an exhaustive study of this problem, under slightly different models. The variations concern the determinism or the randomization of the population protocols. In a sub model, the sensors only communicate with the base station and in another they communicate both between each other and with the base station. According to the different cases, we obtain solutions or prove impossibility results.

Plan of the paper. After some preliminaries to introduce our model (Sec. 2), we will firstly present algorithms in the model where sensors only communicate with the base station (Sec. 3) and secondly in the model where they also communicate with each other (Sec. 4). We conclude in Sec. 5 by summarizing our results in figures and in Sec. 6 by giving final remarks.

2 Motivation and Modelization

Imagine the following scenario : A group of birds (petrels) evolves on an island, carrying on their body a small sensor. Whenever a petrel is close enough to the antenna, its sensor interacts with the antenna which can read the value of the sensor, compute, and then write in the petrel sensor memory.
Depending on the hypothesis, the sensors may or may not interact with each other when two petrels approach close enough.

2.1 Mobile sensor networks

A mobile sensor network is composed of an antenna, and of n undistinguishable mobile sensors (In the sequel we will use the term of petrel, in relation with our motivation example, instead of sensor)
A configuration of the network is given by a and $(p_1, ..., p_n)$ where a is the content of the memory of the antenna, and p_i is the state of the i^{th} petrel.
There are two kinds of events :

- the meeting of petrel number i with the antenna. After that meeting, p_i is changed, according to the protocol, to p'_i, and a to a', depending on (a, p_i) (Note that the transition is independent of i, because petrels are not distinguishable).

- the meeting of petrel number i with petrel number j. After that meeting, p_i and p_j are changed to p'_i and p'_j, depending on (p_i, p_j) (here again, independently of (i, j)).

In the Petrels-To-Antenna-Only model (TA for short), only the first kind of event is possible. i.e. the sensors do not interact with each other.
In the petrels-To-Antenna-And-To-petrels model (TATP for short), both events are possible: sensors do interact with each other.
For deterministic protocols, the last model can be divided into two sub-models, the symmetric (STATP), resp. the asymmetric one (ATATP): When two petrels meet, if their state is the same, they have to, resp.

they don't have to, change to the SAME state.
Note that the asymmetric version can be viewed as probabilistic because there is a need to break the symmetry between the two petrels.

2.2 The problem

The number of petrels is unknown from the antenna which aims at counting them.
That is, we want the $PetrelNumber$ variable in the antenna to be eventually equal to n.
In probabilistic algorithms, we require that this property is obtained with probability 1.
More generally, our algorithms must be self-stabilizing (see [5]), i.e., whatever the initial configuration (but we initialize the antenna), the antenna must give the exact number of petrels in the network (with probability 1, for probabilistic algorithms) within a finite number of steps. This exigence does not allow us to make any assumption on the initial configuration (except for the antenna), or to reset the value of the sensors.

2.3 Executions, Daemons, Fairness, Rounds

Definition 1 (Execution). An execution is an infinite sequence $(C_j)_{j\in\mathbb{N}}$ (where $\mathbb{N}$ denotes the set of non-negative integers) of configurations and an infinite sequence $(e_j)_{j\in\mathbb{N}\setminus\{0\}}$ of events such that C_{j+1} is obtained after e_j occurs on C_j.

The daemon is the imaginary adversary which chooses the initial configuration and that schedules the possible actions at every step. To solve the problem, the daemon must be fair :

Definition 2 (Fairness). An execution is fair if every petrel communicates with the antenna infinitely often, and, in the TATP model, if every two petrels communicate with each other infinitely often.

- A daemon for a deterministic protocol is fair if every execution is fair.
- A daemon for a probabilistic protocol is strongly fair if the execution is fair.
- A daemon for a probabilistic protocol is weakly fair if, the measure of the set of the fair execution is one.

The distinction between weak and strong fairness is of little importance in this paper.

Definition 3 (k-fairness). Let k be an integer. An execution is k-fair, if every petrel communicates with the antenna at least once in every k consecutive events, and, in the TATP model, if every two petrels communicate with each other in every k consecutive events.
A daemon is k-fair if the execution is k-fair.
In this paper when the daemon is k-fair, the value of k is not assumed to be known by the antenna.

Throughout the paper, the daemon is assumed to be at least fair.

Definition 4 (Rounds). A round is a sequence of consecutive events, during which every petrel meets the antenna at least once, and in the TATP model, every two petrels meet each other.
The first round is the shortest round starting from initial configuration, the second round is the shortest round starting from the end of the first round, and so on.

2.4 Initial Conditions

Throughout the paper, we assume that the petrels are arbitrarily initialized, but that an initially value can be chosen for the antenna.
This assumption is justified if one thinks of mobile sensors networks as the petrel population and the antenna.
Note that if both the petrels and the antenna can be initialized , then the problem is obvious, with only one bit per petrel sensor.
Note also that if one can initialize neither the petrels nor the antenna, then there is no protocol to count the petrels (unless the daemon is k-fair, see remark 8 in Sec. 3).

Indeed, assume on the contrary that there is such a protocol. Let the daemon repeat the following: it waits till every petrel has met the antenna and $PetrelNumber = n + 1$ (this will eventually happen), then it holds back one particular petrel. When $PetrelNumber$ is n (this will eventually happen since the configuration is the same as if there were n petrels), the daemon frees the last petrel.
With such a daemon, $PetrelNumber$ will never stabilize so the protocol fails. If the protocol is deterministic, the daemon is fair, if the protocol is probabilistic, it is weakly fair.
To get a the impossibilty result for a strongly fair daemon, proceed by repeating the following: Simulates a strongly fair daemon long enough so that every petrel has met the antenna and the probability that $PetrelNumber = n + 1$ is at least $1/2$. Then hold back one particular petrel, and simulate a strongly fair daemon for n petrels long enough for the probability that $PetrelNumber = n$ to be at least $1/2$.
$PetrelNumber$ will stabilize with probability 0 although the daemon is strongly-fair.

2.5 Memories

We will not make limitation on the memory size of the antenna. On the other hand, we will make more or less strong assumptions on the memory size of the petrel sensors:

Definition 5 (Size of the petrel sensor memories). The memory is infinite if it is unlimited. In particular, it can carry integers as large as needed.
The memory is bounded if an upper-bound P on the number of petrels is known, and if the number of different possible states of the memory is $\alpha(P)$. The protocol will use the knowledge of P.
The memory is finite if the number of different possible states of the memory is α.

3 The petrels-To-Antenna-Only model (TA)

In this section,we present the results for the first model. In this model, the sensors can only communicate with the antenna.

3.1 With infinite memory

In this paragraph, the petrel sensors (and the antenna) are assumed to have an infinite memory, in which case there exists a self-stabilizing deterministic algorithm to solve the problem.
The antenna has registers $(R_k)_{k\in\mathbb{N}}$, where R_k counts the number of sensors with value k already seen.

The algorithm runs as follows : each sensor has a counter initialized in an unspecified value. As soon as a sensor communicates with the antenna, its value is incremented by 1 and the corresponding register of the antenna is incremented. Thus, once the maximum initial value is reached by all the sensors, the value of the registers will constantly be equal to the number of sensors.

Algorithm 1 For Unbounded Memory

```
Memory in the petrel sensors is
        number :integer
Memory at the antenna is
        registers indexed by N, initialized at 0
        PetrelNumber is max{register[i]}
When a petrel with number x approaches the antenna :
        number <- x+1
        R[number] <- R[number]+1
```

Although this algorithm is non observable, it is clearly effective. Note though that the number of rounds before convergence is not bounded by the number of penguins (If there are two penguins, with numbers initialized at 0 and N, it takes N rounds to converge). A less straightforward algorithm, but which converges faster (in about P rounds) is obtained by adding

```
if R[x] > 0 then R[x] --
```

and by taking PetrelNumber = sum{register[i]}.

3.2 With finite or bounded memory

3.2.1 Under a fair daemon

In this paragraph we show that if the daemon only respects fairness, there neither exists a deterministic algorithm nor a probabilistic algorithm making it possible for the antenna to count the number of sensors present.

Proposition 6. *The daemon is supposed to be fair. If the sensors have a finite memory then, there is no deterministic algorithm solving the counting problem.*

Proof : The idea of this proof is to exhibit two executions resulting from two different initial configurations that will appear to be identical for the antenna.
Let us consider an execution E with n sensors $(p_1, \ldots, p_n)$ initialized in $I = (x_1, x_2, \ldots, x_n)$. As the sensors have a finite memory, we easily deduce that there exists a state, denoted by s, that the sensor p_n will visit infinitely often.
Let E' be the following execution with $n+1$ sensors $(p_1, \ldots, p_n, p_{n+1})$. The initial configuration is $I = (x_1, x_2, \ldots, x_n, s)$. The goal of the daemon is to make impossible for the antenna to distinguish p_n from p_{n+1}. In order to do so, it acts as follows :

The daemon holds back the sensor p_{n+1} while letting the others n's evolve as in E. As soon as p_n reaches the state s after having seen the antenna at least once, it replaces it by p_{n+1} that was initialized in state s.
Then it lets p_{n+1} and the other $n-1$'s sensors evolve, while making act p_{n+1} exactly like p_n previously, until the state s is reached by p_{n+1} and the antenna seen at least once by p_{n+1}. once.
Then it replaces it by p_n, and so on.

Therefore, for the antenna, the executions E and E' are identicals. It concludes there are as many sensors in both executions, which is wrong. □
Then, it becomes natural to try to build a probabilistic algorithm in order to break the symmetry. Indeed, the daemon has no control on the random, thus we can hope to beat him. Unfortunately, even in this case, there is no solution :

Proposition 7. *Suppose that the daemon is strongly fair. If the sensors have a finite memory, then it does not exist any probabilistic algorithm solving the counting problem*

Proof : Let us consider a daemon D with n sensors $(p_1, \ldots, p_n)$ initialized in $I = (x_1, x_2, \ldots, x_n)$.
The sensors' memory being finite, for every petrel, in particular for the last one, there is a state s and a positive real number η such that :

$$\mathbb{P}\{p \text{ goes infinitely often in } s\} \geq \eta$$

As we did in the deterministic case, in order to "confuse" the antenna, let the deamon D' proceeds as follows with $n+1$ sensors $(p_1, \ldots, p_n, p_{n+1})$: it puts them in initial configuration $I = (x_1, x_2, \ldots, x_n, s)$.
There is an integer k_1 such that with D, with probability at least $(1-\varepsilon)$, if p_n gets in state s infinitely often, then it gets once in state s during the k_1 next events and every sensor has met the antenna at least once.
The daemon D' holds back the sensor p_{n+1} and for at most k_1 events, lets evolve the other n's as would do daemon D until p_n gets in state s. If k_1 events have been done without p_n getting in state s then D' has lost (note that the daemon may lose either because s does not appear infintely often with D or because the first occurrence of s arrives too late with D). Otherwise D' frees p_{n+1} and holds p_n.
The daemon D' resumes simulating D with p_{n+1} instead of p_n and as in the first step, but with k_1 replaced by k_2 such that the probability is now at least $(1-\frac{\varepsilon}{2})$ instead of $(1-\varepsilon)$. The daemon keeps on with that technique, with k_l for the l^{th} step so that the probability is at least $(1-\frac{\varepsilon}{2^{l-1}})$.
Therefore, D' wins with probability

$$\eta \prod_{i=0}^{l} (1 - \frac{\varepsilon}{2^i}) > 0$$

In this case, the execution is the same as with D so $PetrelNumber$ is eventually equal to n which is wrong. So, the antenna has a non null probability to lose. □

3.3 A k-fair daemon

Under the assumption of fairness, there exists neither deterministic algorithm nor probabilistic algorithm. Thus, we have to reduce the capacities of the daemon.
If we assume the daemon is k-fair, we will get both deterministic and probabilistic solutions.

3.3.1 Deterministic algorithm

Let i_0 be the smallest integer such that $2^{i_0} \geq k$. For every $i \geq i_0$, in the i^{th} loop, the antenna sees all

Algorithm 2 Deterministic, k-fair daemon

```
Memory in the petrel sensors is bit_P : boolean
Memory at the antenna is
        cpt, PetrelNumber : integer
        bit_A : boolean, initialized at 0
The antenna does :
For i from 1 to infinity do
        cpt <- 0
        do 2^i times :
            see arriving one sensor P with bit_P
            if bit_P = bit_A then cpt ++
                                bit_P <- not(bit_P)
        PetrelNumber <- cpt
        bit_A <- not(bit_A)
```

the sensors, so that $Petrelnumber$ is correct from the end of the i_0^{th} loop.
The convergence time is equal to $\sum_{j=1}^{i_0} 2^j = 2^{i_0+1} - 1 \leq 4k$

Remark 8. This algorithm works even if the antenna variables are not initialized but a large initial value of i induces a large convergence time.

The problem of this deterministic algorithm is that it requires an infinite memory of the antenna. We introduce a probabilistic algorithm in order to reduce it. The algorithm is as follows :
Each sensor's value is a pair $n_p = (Id_p, col_p)$ where Id is an identifier and col a color.
Informally, if the antenna sees a new Id it memorizes the Id and the color in the register R_{Id}, if it sees an Id already memorized with the same color it chooses randomly a new color and memorizes it. Algorithm 3 describes the algorithm.

3.3.2 Probabilistic algorithm, k-fair daemon

Proof : Let's C_t be the number of petrels having the same sensor Id than another one after t rounds of the algorithm. It is obvious that the number of non-empty registers in the antenna is greater than or equal to $\bar{P} - Ct$, where $\bar{P}$ is the number of petrels. Thus, after one more round the probability that two petrels with the same sensor have also the same color is lower than $\beta = (1 - \frac{1}{\bar{P}-C_t+1})^k \leq (1 - \frac{1}{\bar{P}})^k$.
The random process C_t is thus bounded by the random walk X_t where :
$X_{t+1} = X_t$ with probability $1 - \beta$.
$X_{t+1} = X_t - 1$ with probability β.
This random walk converges towards 0, this leads to the convergence of C_t towards 0 and to the correctness of the algorithm.
Note that if a bound of the convergence time of the algorithm τ_0 is given by the convergence time of the random walk X_t which is $\tau_0 = \frac{2C_0}{1-\beta}$ which is exponential in k. □
We obtain a worse time of convergence than with the deterministic algorithm but we observe that the antenna requires a finite memory.

Algorithm 3 Probabilistic, k-fair daemon

```
Memory in the petrel sensors is
        number,color: integer
Memory at the antenna is
        registers indexed by N, initialized at empty
        PetrelNumber is card{k | register[k] is non empty}
When a petrel with number n and color c approaches the antenna :
        if R[n] = empty
        then R[n] <- c
        else if R[n] = c
             then color <- random{1..PetrelNumber}
                  R[n] <- color
             else number <- h, where h is the minimum such that R[h] = empty
                  R[number] <- c
```

4 The Petrels-To-Antenna-And-To-petrels model (TATP)

We recall that P is an upper bound of the number of petrels and $\alpha(P)$ is the number of the different possible states of the memory. In a first section we introduce deterministic algorithms solving the counting problem. Then, in a second part, we get interested in the lowest value $\alpha(P)$ may get.

4.1 Bounded memory, algorithms

Proposition 9. *There are deterministic solutions, with $\alpha(P) \geq P$, to the counting problem.*

We are going to exhibit different algorithms. The two first ones concern the ATATP model and the third one the STATP model. It is interesting to note that different memory capacities are required in the two models.

4.1.1 The ATATP model We propose two algorithms :

- The first one with $\alpha(P) = P + 1$, converges in three rounds.
- The second one with $\alpha(P) = P$, converges in $P + 1$ rounds.

Here is an asymmetric algorithm with $\alpha(P) = P + 1$, converging in three rounds:

Algorithm 4 Deterministic asymmetric algorithm with $\alpha(P) = P + 1$

```
Memory in the petrel sensors is
        number :integer in [0..P]
Memory at the antenna is
        T array [1..P] of boolean, initialized at 0 everywhere
        PetrelNumber is the number of i such that T[i]=1
When a petrel with number x approaches the antenna :
        if x = 0
        then let y be an integer such that T[y]=0
                T[y] <- 1
                number <- y
        else T[x] <- 1
When two petrels meet :
        If their numbers are the same
        then the number of one petrel becomes 0
```

Sketch of proof of correction :
At every moment, if $T[x] = 1$, then there exists a petrel with number x.

¿From the end of the first round, the converse is also true.
¿From the end of the second round, no two petrels have the same number (but maybe 0).
¿From the end of the third round, petrels have distinct positive numbers.

Let's now introduce an asymmetric algorithm with $\alpha(P) = P$, converging in $P + 1$ rounds:

Algorithm 5 Deterministic asymmetric algorithm with $\alpha(P) = P$

```
Memory in the petrel sensors is
        number :integer in [1..P]
Memory at the antenna is
        T array [1..P] of boolean, initialized at 0 everywhere
        PetrelNumber is the number of i such that T[i]=1
When a petrel with number x approaches the antenna :
        T[x] <- 1
When two petrels meet :
        If their numbers are the same integer x
        then the number of one petrel becomes x+1 mod P
```

Sketch of proof of correction :
After k rounds, if there is no petrel with number x, then, there is a most one petrel with id y, for every y in $\{x + 1, x + 2, x + 3, ..., x + k\}$ (numbers are taken mod P of course).
After P rounds, petrels have distinct numbers that they will keep.
After $P + 1$ round, $T[x] = 1$, iff there exists a petrel with number x.

4.1.2 The STATP model The following symmetric algorithm with $\alpha(P) = 4P$ converges in three rounds:

Algorithm 6 Deterministic symmetric algorithm with $\alpha(P) = 4P$

```
Memory in the petrel sensors is
        number :integer in [1..2P]
        Intention : (Keep,GiveUp)
Memory at the antenna is
        T array [1..2P] of (Free,Taken,GivenUp),
                initialized at Free everywhere
        PetrelNumber is the number of i such that T[i]=Taken
When a petrel with number x approaches the antenna :
        Depending on Intention :
        Keep :   T[x] <- Taken  /* even if T[x] was GivenUp */
        GiveUp : T[x] <- GivenUp
                 find a y such that T[y] = Free
                 T[y] <- Taken
                 number <- y
                 Intention <- Keep
When two petrels meet :
        If     their numbers are the same integer x
           and their both intentions are Keep
        Then their both intentions change to GiveUp
```

Sketch of proof of correction :
The antenna never gives twice the same number to a petrel.

If at some point in the execution, $T[x] = GivenUp$, then there was a petrel carrying that number x at the beginning of the algorithm (its intention could be then either Keep or GiveUp). Thus, there are at most P different numbers which get given up during the execution.
If at some point on the execution, $T[i] = 1$, then either there is a petrel with $number = i$, or i is one of the numbers which get given up during the execution.
When the antenna needs to find a free y, then there are at most P numbers which are to be given up at some point in the algorithm. There are at most $P - 1$ numbers with $T[i] = 1$ and which are never given up during the execution. Thus there is at least one free y that the antenna finds and can give to the petrel.

- From the end of the first round :
  - If $T[x] = 0$ at some instant,then there is no petrel with that number at that same instant. And thus, when the antenna gives a number to a petrel, this petrel will keep it forever.
  - There is at most two petrels with the same number and with Intention to Keep.
- At the end of the second round,
  - Either there is no petrel carrying x, whatever the intention.
  - Or there is only one petrel carrying x, and its intention is $Keep$.
  - Or there are two petrels carrying x, and both their intention is $GiveUp$.
- From the end of the third round,
  - Petrels have different numbers and every $Intention$ is $Keep$, and $T[x] = Taken$, iff there is a petrel carrying x.

4.2 Bounded memory, minimum value for $\alpha(P)$

We prove in this section there does not exist asymmetric algorithms with $\alpha(P) \leq P - 1$.
The non-existence of algorithms with $\alpha(P) \leq P - 2$ is much easier to prove than the non-existence of algorithms with $\alpha(P) = P - 1$. So let us start with the easier case:

Proposition 10. *There is no deterministic solution, with $\alpha(P) \leq P - 2$, to the counting problem.*

Proof :
Assume that there is a solution. Consider an execution E with $P-1$ sensors $(p_1, \ldots, p_{P-1})$ initialized in the states $(x_1, \ldots, x_{P-1})$.
There is a state y and two petrels p and p' such that infinitely often, p and p' will be simultaneously in state y.
Now, as a daemon, perform the following execution E' with P sensors:
Initialize them in $(x_1, \ldots, x_{P-1}, y)$, then repeat the following:

- Hold back petrel p_P and proceed as in E until every petrel but p_P has met each other petrel but p_P, and p and p' are in state y.
- Free p_P, hold back p, proceed as in E with p_P instead of p until p_P has met every other petrel (but p), and p_P and p' are again in state y.
- Free p, hold back p', proceed as in E with p_P instead of p' until p_P has met p, and p_P and p are again in state y.

The daemon is fair, and from the point of view of the antenna, E and E' are identical, thus in E', $PetrelNumber$ will stabilize to $P - 1$, as in E, which is a wrong result. This is a contradiction.
□

We are now going to look to the case where $\alpha(P) = P - 1$.

Proposition 11. *There is no deterministic solution with, $\alpha(P) = P - 1$, to the counting problem.*

Proof :
Assume on the opposite that there is such a solution.
Consider an execution E with $P - 1$ sensors $(p_1, \ldots, p_{P-1})$ initialized in the states $(x_1, \ldots, x_{P-1})$.
We split the proof in two cases :

If there is a state y and two petrels p and p' such that infinitely often, p and p' are simultaneously in state y, then one can conclude as in the previous proof, so we can assume from now on it is not the case. This implies that eventually, say from instant T, all petrels have distinct states.
This means first that, in E from T, the antenna never changes the state of a petrel it meets.
Second, every petrel eventually meets every other petrel, and what happens in that case depends only on their current state (since the protocol has no room to remember whatsoever). Thus the rule when two petrels with different states meet must be that they keep their current state (or exchange them, which is of little effect).
Thus the protocol rules for meeting petrels are such that the states can change only if the meeting petrels are in the SAME state.

Lemma 12. *There is a state y and a finite piece of execution E_{KL} with P petrels, starting with two petrels in state y and one petrel in each other state, finishing in the same configuration, during which petrels do not meet the antenna, and whose first event is the meeting of the two petrels in state y.*

We postpone the proof of that key lemma.
Now, as a daemon, proceed as follows with P petrels:
Put them in initial states $(x_1, \ldots, x_{P-1}, y)$.
Hold the last petrel, and proceed as in E until T, then repeat the following:

- Free the held on petrel, make every two petrels of different states meet twice in a row (so that if they exchange their states, they get their former states back).
- Proceed as in E_{KL}.
- Hold on one of the two petrels in state y, not the one you have held just before.
- resume the simulation of E long enough for every petrel (but the held one) to meet the antenna.

Here again, the daemon is fair, and from the point of view of the antenna, E and E' are identical, thus in E', $PetrelNumber$ will stabilize to $P-1$, as in E, which is a wrong result. This is a contradiction. □
It remains now to prove the key lemma :
Let us introduce two kinds of vectors, the first one for representing the states of all the sensors at a given time, the second one to represent the effect of the meeting of petrels.

Definition 13. The *vector of configuration* V_C of configuration C is the vector in $\mathbb{N}^{P-1}$ whose i^{th} coordinate is the number of sensors in the i^{th} state s_i.

For example, the vector $(1, 1 \ldots, 1)$ indicates there are $P-1$ petrels, and that there is one sensor in each different state. This is a vector of configuration for E.
The Key Lemma will deal with $(1, 1 \ldots, 1) + \mathbb{1}_y$, where $\mathbb{1}_x$ denotes the vector which is 0 everywhere but in the coordinate of x, where it is 1.
For each state x, let us define $y(x)$ and $z(x)$ to be the states that two petrels sensors get when they meet while both in state x.

Definition 14. The *vector of variation* V_x of state x is $\mathbb{1}_{y(x)} + \mathbb{1}_{z(x)} - 2\mathbb{1}_x$.

The i^{th} coordinate of V_x represents the variation of the number of sensors in state s_i when two petrels in state x meet, and indeed, if, from a configuration represented by V, two petrels in state x meet, the new configuration is represented by $V + V_x$.
Such a vector could be $(0, \ldots, 0, +1, -2, 0, +1, 0, \ldots, 0)$ the -2 meaning that the two states e_i disappear and give rise to the states e_{i+2} and e_{i-1}. Note though that the values of such vectors are in $\{-2, -1, 0, 1, 2\}$ depending on whether x, $y(x)$ and $z(x)$ are distinct. The vector V_x will even be the null vector if two meeting petrels in state x do not change their state.
Note that the vectors of variation are in the $(P-2)$-dimensional subspace Z of $\mathbb{N}^{P-1}$ defined to be the set of vectors whose sum of the coefficients is null.

We claim first that there is a non-null linear combination of the vectors of variations, with non-negative integer coefficients, which is null.

Note that it is easy to prove there is a non-null linear combination which is null, since there are $P-1$ such

vectors, and they are all in the $(P-2)$-dimensional subspace Z, but this is of little help because it gives a combination with potentially negative coefficients (to get integer coefficient is not a main problem). Using that idea is a dead-end.
To prove the claim, start with P petrels and repeat making two petrels in the same state meet each other (you will always find two such petrels). The vectors of configuration you will get will stay in $Y = \{(q_j)_{1\leq j<P} | q_j \in \mathbb{N}, \sum_j q_j = P\}$ which is finite. So if you let long enough petrels with same state meet, you will encounter twice the same configuration. The set of meetings between the two appearances of that configuration gives you the desired combination.
More formally: define $(V_{y,i})_{0\leq i} \in Y^{\mathbb{N}}$ and $(V_{w,i})_{1\leq i} \in Z^{\mathbb{N}}$ by induction as follows:
$V_{y,0} = (2, 1, 1, 1, \ldots, 1)$.
Once $V_{y,i}$ is defined, find a coefficient x of $V_{y,i}$ which is at least 2 (there is such a coefficient), then define $V_{y,i+1} = V_{y,i} + V_x$ and $V_{w,i+1} = V_x$. It is easy to check that $V_{y,i+1}$ will be in Y.
Since Y is finite, there are two integers i_1 and i_2, with $0 \leq i_1 < i_2 \leq card(Y)$ such that $V_{y,i_1} = V_{y,i_2}$. Then $\sum_{i_1<i\leq i_2} V_{w,i}$ fulfills the requirement since $V_{y,i_2} = V_{y,i_1} + \sum_{i_1<i\leq i_2} V_{w,i}$.
That first claim is proved.
Let $\sum_x \beta_x V_x$ be such a combination (So, $\forall x, \beta_x \in \mathbb{N}$, $\exists x, \beta_x > 0$, and $\sum_x \beta_x V_x = (0, 0, \ldots, 0)$). For the sake of simplicity, let us assume that our combination minimizes $\sum_x \beta_x$.
Let H be the multi set of vectors of variations where each V_x appears β_x times.

Our second claim is that there is an index y and an ordering $(h_1, h_2, ..., h_{card(H)})$ of the elements in H, such that $h_1 = V_y$, and for every $i \in [1, card(H)]$, the ${h_i}^{th}$ coordinate of $Z_i = (1, 1, \ldots, 1) + \mathbb{1}_y + \sum_{j<i} h_j$ is 2 or more, and no coordinate of Z_i is negative.
Proof of the second claim:
Let y be an index such that $\beta_y > 0$.
We build the h_i by induction on i : Let $h_1 = V_y$
Assume the h_j's have been built up to $j = i - 1$, let us build h_i, for some $i \in [2, card(H)]$:
Let $Z_i = (1, 1, \ldots, 1) + \mathbb{1}_y + \sum_{j<i} h_j$. Since it is in Y, there is an index x such that $(Z_i)_x \geq 2$. We may assume that $x \neq y$ or ($x = y$ and $(Z_i)_x \geq 3$) (indeed, otherwise, $Z_i = (1, 1, \ldots, 1) + \mathbb{1}_y$, which implies that $\sum_{j<i} h_j = 0$, which contradicts the minimality of $\sum_x \beta_x$ in our combination).
Thus $\sum_{j<i}(h_j)_x > 0$, but since $\sum_{h\in H} h_x = 0$, it means that there is an element in H, not taken yet, whose x^{th} coordinate is negative. This element is V_x for it is the only vector of variations whose x^{th} may be negative. Let $h_i = V_x$.
The built sequence $(h_1, h_2, ..., h_{card(H)})$ satisfies the requirement, so the second claim is proved.

The E_{KL} execution is the following:
Start with P petrels, two of them in state y, and one of them in each other state.
For i from 1 to $card(H)$, make two petrels in state x_i meet, where x_i is the state such that $V_{x_i} = h_i$ (there are two such petrels thanks to the propriety on Z_i is the second claim) □
Notes on the key lemma :
(1) A little adaptation of the proof shows the following stronger result: Assume there is a population of $P - 1$ petrels with one petrel in each state. Introduce a new petrel in state y, and let petrels meet fairly. Wait till the two petrels in state y meet, and then, observe the petrels and if a petrel in state y re-appears, grab it back. The result is that you will indeed see a petrel in state y re-appear, plus eventually, the remaining $P - 1$ petrels will be one in each state.

(2) The upper-bound on the length of E_{KL} given by the proof is $card\{\{(q_j)_{1\leq j<P} | q_j \in N, \sum_j q_j = P\}$ which is exponential in P. One can wonder if it has to be large, or if there is such an execution E_{KL} of size polynomial in P. The answer is that it might be indeed exponential. Consider the set of states $[0, P-1]$. Take $y = 0$ (that is, start, with two petrels in state 0, and one in each state $i \in [1, P-1]$), and let the protocol be that when two petrels in state i meet, one of them gets in state 0, the other one gets in state $(i + 1)$ mod P.

5 Resume

The TA model

model \ memory	**Finite**	**Bounded**	**Bounded,k-fair daemon**	**Unbounded**
deterministic	impossible	impossible	Algorithm 2	Algorithm 1
convergence time			4k events	depends on initialization
probabilistic	impossible	impossible	Algorithm 3	unneeded
convergence time			exponential in k	

The TATP model

model \ memory	**Finite**	**Bounded,**$\alpha(P) < P$	**Bounded,**$\alpha(P) \geq P$
symetric deterministic	impossible	impossible	Algorithm 6
convergence time			$\alpha(P) = 4P$, 3 rounds
asymetric deterministic	impossible	impossible	Algorithm 4 or 5
convergence time			$\alpha(P) = P+1$, 3 rounds $\alpha(P) = P$, P+1 rounds

6 Final Remarks

In this article, we have studied the problem of self-stabilizing counting in different models of mobile sensor networks. We designed different algorithms depending on the communication model and the class of daemon. We also gave some proof of impossibility. In the cases where no deterministic (symmetric) solutions exists, we proposed probabilistic solutions. The knowledge of the size of a population is at the basis of the solutions of more complex problems, in particular when different types of population are present.
An interesting perspective could be to modelize the movement of the sensors, by random processes for example, in order to improve our algorithms and to get better bounds for the convergence time.

7 References

[1] D. Angluin, J. Aspnes, M.J. Fischer, and H. Jiang. Self-stabilizing population protocols. In *Proc. of 9th Int. Conference on Principle of Distributed Systems OPODIS 2005*, pages 103–117. LNCS 3974, 2005.

[2] D. Angluin, Eisenstat D., and E. Ruppert. The computational power of population protocols. In *Proc. of 25 Annual Symposium on Principle of Distributed Computing PODC 2006*, 2006.

[3] D. Angluin, Eisenstat D., and E. Ruppert. Fast computation by population protocols. In LNCS, editor, *Proc. of 20th International Symposium on Distributed Computing DISC 2006*, pages 61–75, 2006.

[4] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. In *DCOSS*, pages 51–66, 2006.

[5] S. Dolev. *Self-Stabilization*. The MIT Press, 2000.

[6] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring, 2002.

[7] K. Martinez, J.K. Hart, and R. Ong. Environmental sensor networks. *IEEE Computer*, 37(8):50–56, 2004.

[8] C. Ulmer, Yalamanchili S., and L. Alkalai. Wireless distributed sensor networks for in-situ exploration of mars. In Georgia Institute of Technology and California Institute of Technology, editors, *Technical Report*, 2003.

[9] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A wireless sensor network for structural monitoring. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 13–24, New York, NY, USA, 2004. ACM Press.