Self-Stabilizing Counting in Mobile Sensor Networks with a Base Station

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

Univ. Paris Sud, LRI, UMR 8623, Orsay, F-91405;
CNRS, Orsay, F-91405
Contact person: J. Clement, e-mail: clement@lri.fr

Abstract. Distributed computing must adapt its techniques to networks of mobile agents. Indeed, we are facing 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 [4, 3, 2, 1]) 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 base station 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 Best Student Paper Award

1 Introduction

Habitat and environmental monitoring represents a class of sensor network applications with enormous potential benefits both for scientific communities and for 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 [9]), use of sensors is considered in Mars exploration (see [10]) or for detecting the effect of the wind or of an earthquake on a building (see [11]). A sensor network has been deployed on Great Duck Island (see [8]) 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 [1], 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, is provided with more computing resources.
The codes in the base station and in the sensors define 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 consequently, 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 some 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 this model assumes that the code is immutable, e.g, 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 [4] and in a different framework in [6].
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 a 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.

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 base station, its sensor interacts with the base station, 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 with a base station

A mobile sensor network is composed of an base station, 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.)
The network configuration considers the memory content of the base station, a, and the petrels' state, p_i. We denote the network configuration by $(a, p_1, ..., p_n)$ where p_i is the state of the i^{th} petrel. There are two kinds of **events** :
• the meeting of petrel number i with the base station. 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 Sensors-To-base-station-Only model (TB for short), only the first kind of event is possible. i.e. the sensors do not interact with each other.
In the petrels-To-Base-station-and-To-Petrels model (TBTP for short), both events are possible: sensors do interact with each other. For deterministic protocols, the last model can be divided into two submodels, the symmetric (STBTP), resp. the asymmetric one (ATBTP): When two petrels meet, if their state is the same, they have to, resp. they don't have to, change to the SAME state. A probabilistic algorithm can use coin flips and perform an election between meeting petrels to simulate the Asymmetric TBTP model.

2.2 The problem

The number of petrels is unknown from the base station, which aims at counting them.
We want that, eventually, the $PetrelNumber$ variable in the base station is and remains equal to n.

In probabilistic algorithms (we consider non-oblivious daemons, that is, they can decide what is the next event depending on previous results of coin flips), we require that this property is obtained with probability 1.
More generally, our algorithms must be self-stabilizing (see [7]), i.e., whatever the initial configuration (but we initialize the base station), the base station must give the exact number of petrels in the network (with probability 1, for probabilistic algorithms) within a finite number of steps. This requirement does not allow us to make any assumption on the initial configuration (except for the base station), 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 that 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 base station infinitely often, and, in the TBTP model, if every two petrels communicate with each other infinitely often. (Note that this fairness is weaker than the one used by Angluin et al., which says that a configuration that is reachable infinitely often is eventually reached)*
•A daemon for a deterministic protocol is fair if every execution is fair.
•A daemon for a probabilistic protocol is strongly fair if every execution is fair and it is weakly fair if the measure of the set of the fair executions 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 base station at least once in every k consecutive events, and, in the TBTP 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 base station.

Throughout the paper, the daemon is assumed to be fair, unless it is explicitly assumed to be k-fair.

Definition 4 (Oblivious). *For probabilistic algorithms, a daemon is non-oblivious if the decision of what is the next event can depend on the result of previous coin flips. An oblivious daemon could be able to decide at the start of the execution what the whole sequence of events will be.*

Definition 5 (Rounds). *A round is a sequence of consecutive events, during which every petrel meets the base station at least once, and in the TBTP 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 initial value can be chosen for the base station. This assumption is justified if one thinks of mobile sensors networks as the petrel population and the base station. The existence of a base station and the possibility to initialize it are the main differences between our model and classical sensor networks.
Note that if both the petrels and the base station 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 base station, then there is no protocol to count the petrels (unless the daemon is k-fair, see remark 1 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 base station and $PetrelNumber = n$ (this will eventually happen), then it holds back one particular petrel. When $PetrelNumber$ is $n-1$ (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.
It can also be proved that there does not exist any algorithm under a strongly-fair daemon $PetrelNumber$ 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 base station. (Note: The codes will often use "infinite" arrays (indexed by integers), but only a finite number of register will contain non-0 values. Of course, in practice, arrays will have to be replaced by data structures to keep only the non-0 registers.)
On the other hand, we will make more or less strong assumptions on the memory size of the petrel sensors:

Definition 6 (Size of the petrel sensor memories). *The memory is* **infinite** *if it is unlimited. In particular, it can carry integers as large as needed, which can drift, that is, which can tend to infinity as times passes by. (this has a practical application only if the drift is slow and there are enough bits in the sensors to carry "large"integers). 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 $\alpha(P)$ for some function α. The protocol may use the knowledge of P. The memory is* **finite** *if the number of different possible states of the memory is a constant α.*

3 The petrels-To-Base-station-only model (TB)

In this section, the sensors can only communicate with the base station. People acquainted with classical sensor networks may question the point of such a model. There are two justifications for looking for solutions in that model :

1. Sensors are meant to be small. To implement that model, sensors only need to carry a device so that the base station can read and write in their memories. All the code can be implemented in the base station.
2. The decision to run such algorithms can be made by just changing the code in the base station. This is doable even if the sensors are already away. For example, if an observation made of the petrels, given you the idea to count something new and not forecast, you can use a TB algorithm.

3.1 With infinite memory

In this subsection, we assume that the petrel sensors (and the base station) have an infinite memory. In this case, there exist self-stabilizing deterministic algorithms to solve the problem.
The way the first algorithm works is simple. The drift of integers is fast, and convergence is obtained after two rounds. The second one is a little tricky. The drift is slow, but it converges in about P (i.e. the number of petrels) rounds.

Algorithm 1 For Unbounded Memory

```
variables
        [each petrel] number :integer
        [base station] R : array[integers] of booleans, initialized at 0
        [base station] PetrelNumber to maintain as cardinal{i | R[i]=1}
        [base station] LargestNumber : integer initialized at 0
When a petrel p approaches the base station :
        if R[number_p] = 1 then R[number_p] <- 0
        number_p <- LargestNumber
        R[LargestNumber] <- 1
        LargestNumber ++
```

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 base station to count the number of sensors present.

Algorithm 2 For Unbounded Memory

```
variables
        [each petrel] number :integer
        [base station] R : array[integers] of booleans, initialized at 0
        [base station] PetrelNumber to maintain as sum{R[i]}
When a petrel p approaches the base station :
        if R[number_p] > 1 then R[number_p] --
        number_p ++
        R[number_p] ++
```

Proposition 1. *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 base station.The proof is analog to the one of proposition 10, but details are in the full version [5]
□
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 2. *Suppose that the daemon is strongly fair and non-oblivious. If the sensors have a finite memory, then there 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 p, in particular for the last one, there is a state s and a positive real number η such as : $\mathbb{P}\{p \text{ goes infinitely often in } s\} \geq \eta$
In order to "confuse" the base station, let the deamon D' proceeds as follow with n+1 sensors $(p_1, \ldots, p_n, p_{n+1})$: it puts them in the 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 base station 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, from the point of view of the base station, the execution is indistinguishable from D, so $PetrelNumber$ is eventually equal to n which is wrong. So, the base station has a non null probability to lose. □

Note that the proofs work with no assumption on n (the number of petrels) which may be equal to 1. Thus the impossibility is proved both for finite and bounded memories.

3.3 A k-fair daemon

Under the assumption of fairness, there exists neither a deterministic algorithm nor a 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

The algorithm 3 is given below.
The convergence time is less than $8k$. The reader may find details in the full version [5]

Algorithm 3 Deterministic, k-fair daemon

```
variables
      [each petrel] bit : boolean
        [base station] i, cpt, PetrelNumber : integers
        [base station] bit_A : boolean, initialized at 0
The base station does :
For i from 0 to infinity do
        cpt <- 0
        do 2^i times :
            wait till a petrel p approaches
            if bit_p = bit_A then cpt ++
                                 bit_p <- not(bit_p)
        PetrelNumber <- cpt
        bit_A <- not(bit_A)
```

Remark 1. This algorithm works even if the base station variables are not initialized but a large initial value of i induces a large convergence time.

This deterministic algorithm requires an infinite memory of the base station, due to the drift of 2^i (and of i). This can be avoided by the following probabilistic algorithm.

3.3.2 Probabilistic algorithm, k-fair daemon or oblivious daemon

The algorithm is as follows :

Algorithm 4 Probabilistic, k-fair daemon

```
variables
        [each petrel] number, color : integer
        [base station] R : array[integers] of [0...2]/* 0 stands for empty, others for colors */
                    initialized at empty
        [base station] PetrelNumber to maintain as card{j | R[j] is non empty}
When a petrel p approaches the base station :
         h <- the minimum integer such that R[h] = empty
         if R[number_p] <> color_P /* including if one of them is 0 */
         then   number_p <- h
         else if h<number_p
              then R[number_p]<-0
                 number_p<-h
        color_p <- random{1..2}
        R[number_p] <- color_p
```

The proof of convergence is in the full version [5] We obtain a worse time of convergence (possibly exponential) than with the deterministic algorithm but we observe that the base station requires a finite memory.

4 The petrels-To-Base-station-And-To-Petrels model (TBTP)

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 3. *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 ATBTP model and the third one the STBTP model. It is interesting to note that we need more memory in the STBTP model. The question remains open to know what is the minimal memory required in the symmetric case, and if it really needs to be larger than in the asymmetric case. Explanations of the algorithms are in the full version [5]

4.1.1 The ATBTP 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.

Algorithm 5 Deterministic asymmetric algorithm with $\alpha(P) = P + 1$

```
variables
        [each petrel] number :integer in [0..P]
        [base station] T : array [1..P] of boolean, initialized at 0 everywhere
        [base station] PetrelNumber to maintain as cardinal{i | T[i]=1 }
When a petrel p approaches the base station :
        if number_p = 0
        then    number_p <- an integer y such that T[y]=0
                T[number_p] <- 1
        else T[number_p] <- 1
When two petrels meet :
        If their numbers are the same
        then the number of one petrel becomes 0
```

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

```
variables
        [each petrel]  number :integer in [1..P]
        [base station] T : array [1..P] of boolean, initialized at 0 everywhere
        [base station] PetrelNumber to maintain as cardinal{i | T[i]=1 }
When a petrel p approaches the base station :
        T[number_p] <- 1
When two petrels meet :
        If their numbers are the same integer x
        then the number of one petrel becomes x+1 mod P
```

4.1.2 The STBTP model

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

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

Proof :

Algorithm 7 Deterministic symmetric algorithm with $\alpha(P) = 4P$

```
variables
        [each petrel] number :integer in [1..2P]
        [each petrel] Intention : (Keep,GiveUp)
        [base station] T array [1..2P] of (Free,Taken,GivenUp), initialized at Free everywhere
        [base station] PetrelNumber to maintain as cardinal{i | T[i]=Taken }
When a petrel p approaches the base station :
        Depending on Intention_p :
        Keep :   T[number_p] <- Taken  /* even if T[number_p] was GivenUp */
        GiveUp : T[number_p] <- GivenUp
                 number_p <- a y such that T[y] = Free
                 T[number_p] <- Taken
                 Intention_p <- 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
```

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 base station, 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 5. *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 $(x_1, \ldots, x_{P-1})$.
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 base station never changes the state of a petrel it meets. Second, 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 1. *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 base station, and whose first event is the meeting of the two petrels in state y.*

The end of the proof is analog to the proof of proposition 10, but the reader may find the entire proof in the full version [5]

□

It remains now to prove the key lemma (the detailled proof is in the full version [5]) :
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 7. *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 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 8. *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 $V + V_x$.
We claim first that there is a non-null linear combination of the vectors of variations, with non-negative integer coefficients, which is null.

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 wanted 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 $\delta(i)^{th}$ 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 (where $delta(i)$ is the index such that $h(i) = V_{\delta(i)}$)
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 $Z_i|_x \geq 2$ (where $M|_v$ denotes the v^{th} coordinate of M). 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) □
Note on the key lemma :
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 TB model

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

The TBTP model

model \ memory	**Finite**	**Bounded,**$\alpha(P) < P$	**Bounded,**$\alpha(P) \geq P$
symmetric deterministic	impossible	impossible	Algorithm 7
convergence time			$\alpha(P) = 4P$, 3 rounds
asymmetric deterministic	impossible	impossible	Algorithm 5 or 6
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 exist, 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 model 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.

Acknowledgments This work was supported by grants from Region Ile-de-France.

7 References

[1] D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *Proc. of 23 Annual Symposium on Principle of Distributed Computing PODC 2004*, 2004.

[2] D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. In LNCS, editor, *Proc. of 20th International Symposium on Distributed Computing DISC 2006*, pages 61–75, 2006.

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

[4] 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.

[5] J. Beauquier, J. Clement, S. Messika, L. Rosaz, and B. Rozoy. Self-stabilizing counting in mobile sensor networks. In *Technical Report L.R.I , number 1470*, March 2007.

[6] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. When birds die: Making population protocols fault-tolerant. In *DCOSS*, pages 51–66, 2006.

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

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

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

[10] C. Ulmer, S. Yalamanchili, 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.

[11] N. Xu, S. Rangwala, K. Kant Chintalapudi, D. Ganesan, A.Broad, R. Govindan, and D. 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.