A fault-tolerant communication mechanism for cooperative robots

Joyce El Haddad* Serge Haddad

Lamsade Laboratory, CNRS-UMR 7024
University of Paris Dauphine
Place du Maréchal De Lattre de Tassigny
75775 Paris Cedex 16, France
Tel : +33.1.44.05.41.20
Fax : +33.1.44.05.40.91
{elhaddad,haddad}@lamsade.dauphine.fr

Key words : multi-robots systems, fault tolerance, self-stabilization, Petri nets, Markov chains.

*Corresponding author.

A fault-tolerant communication mechanism for cooperative robots

Joyce El Haddad Serge Haddad

Lamsade Laboratory, CNRS-UMR7024, University of Paris Dauphine

Abstract

Operations in unpredictable environments require coordinating teams of robots. This coordination implies peer-to-peer communication between the team's robots, resource allocation, and coordination. We address here the problem of autonomous robots which alternate between execution of individual tasks and peer-to-peer communication. Each robot keeps in its permanent memory a set of locations where it can meet some of the other robots. The proposed protocol is constructed by two layered modules (sub-algorithms: a self-stabilizing scheduling and a construction of a minimum-hop path forest). The first self-stabilizing algorithm solves the management of visits to these locations ensuring that after the stabilizing phase, every visit to a location will lead to a communication. We model the untimed behaviour of a robot by a Petri net and the timed behaviour by an (infinite) Discrete Time Markov Chain. Theoretical results in this area are then combined in order to establish the proof of the algorithm. The second self-stabilizing algorithm computes the minimum-hop path between a specific robot's location and the locations of all the other robots of the system in order to implement routing.

1 Introduction

As robots become more on more autonomous and sophisticated, they are increasingly used for complex tasks. For some of them, a team of robots is needed to achieve some goal in too dangerous or undesirable areas for humans. Such tasks could include maintaining of nuclear reactors, as well as participating in search and rescue missions. For instance, it would be much better to deploy several robots for the maintenance of nuclear reactors where one robot identifies the faulty component and another one follows to replace it with a new one.

For many other tasks, not only multiple robots are necessary, but explicit coordination amongst them is imperative. There has been an increasing research interest in coordination and cooperation as a step towards achieving systems of multiple mobile robots engaged in a collective behaviour (Cao *et al.* 1997, Luzeaux

2000, Defago and Konagaya 2002). Furthermore, by working simultaneously, each robot of the team can achieve its own task. One such application domain is the exploration of the surface of Mars. In fact, Sojourner, a US rover, has already landed on Mars in 1997, and two more NASA rovers are planned to land on Mars in 2004, each having much more autonomy and capabilities than Sojourner. Their mission is to perform different individual tasks, such as picking up rocks, and to jointly achieve common objectives, such as communicating their measures of the atmospheric pressure and temperature.

An important reason for using several small rovers instead of a single large rover during a Mars mission is to increase efficiency. Indeed, several small rovers can provide a greater scientific return than a single large rover by exploring a greater surface area. Another important reason is cost reduction, smaller rovers are less costly to build and to land on Mars. Finally, such an organisation is fault tolerant since the failure of a single rover will not incapacitate the whole mission. In such an environment, there are two kinds of protocols to design: a synchronization protocol between neighboring robots in order to establish (temporary) point-to-point communications and a routing protocol in order to exchange packets between two robots (and in particular distant ones). Although there has been significant work related to cooperating robots (Park and Corson 1997, Hu *et al.* 1998, Prencipe 2001), very few authors approaches incorporate fault tolerance mechanisms. Our research focuses on the introduction of self-stabilization as an efficient property that makes the system tolerant to faults and takes into account the limited resources of the robots in term of processor, memory and energy. Roughly speaking, a self-stabilizing protocol is designed to recover from an unsafe state caused by a fault to a safe state by itself. The study of self-stabilization started with the fundamental paper of Dijkstra (1974). Following this paper, a great amount of works has been done in this area (Schneider 1993, Dolev 2000). However, in the presence of mobility and dynamic changes, existing communication protocols, meant for self-stabilizing networks, are no more appropriate.

In this work, we first describe, in Section 2, a scheduling protocol and then we use Petri nets theory to give a new proof of its correctness. Next in Section 3, we present a transformation of the above protocol into a uniform self-stabilizing scheduling protocol that manages visits to the locations ensuring that after the stabilizing phase, every visit to a location will lead to a communication. We model the executions of the resulting protocol by a discrete time Markov chain. Based on this stochastic semantic, we prove the correctness of our protocol in Section 4. In Section 5, we present how to compute the minimum-hop path between a specific robot's location and any location of another robot of the system in order to implement routing. Finally, in Section 6, we give its correctness proof.

2 Scheduling protocol

In this section, we present the non self-stabilizing scheduling protocol of Bracka *et al.* (2003) for a robotic network. Let us first describe the system model and give some assumptions

2.1 The system model

1. The system consists of a finite set of m robots denoted by $\{r_1, \ldots, r_m\}$ and an ordered set of N meeting points, henceforth *locations*. In this section, robots are *anonymous* in the sense that, they do not know their identifiers (the subscript i of r_i is used for convenience of explanation), they cannot be distinguished by their appearances, and they all use the same algorithm for determining the next position. Each location has its own distinct identity, that does not change during the protocol execution. We will denote this set by $\{l_1, \ldots, l_N\}$ in increasing order. A pair of robots is associated with each location where a temporary communication can be established, if both of the robots are present.

2. Each robot r_i has hardwired an array of the locations where it can go. This array, of length n_i, is sorted in increasing order of the identifiers and each entry stores the identifier of the jth location of the robot r_i given by the function $f(i, j)$ for $1 \leq i \leq m$ and $0 \leq j \leq n_i - 1$.

3. Between any pair of robots r_i and $r_{i'}$, there is a sequence of robots $r_i = r_{i_0}, r_{i_1}, \ldots, r_{i_K} = r_{i'}$ such that for all $0 \leq k < K$, r_{i_k} and $r_{i_{k+1}}$ share a location. This hypothesis ensures that there is a (potential) global connectivity between robots.

The aim of the protocol is to schedule the visits of the locations for each robot in such a way that every location is infinitely often visited. The obvious requirement is that a robot cannot leave a location without establishing a communication with the other robot, henceforth called *peer*, associated with this location. In the scheduling algorithm of Bracka *et al.* (2003) each robot infinitely visits its locations following the order of its array starting from the first one. Bracka *et al.* developed a specific, yet rather lengthy, proof that no (partial or global) deadlock can occur. With the help of Petri net theory, we gave a shorter and simpler proof of the algorithm (El Haddad and Haddad 2003). We recall this proof since it will be the basis of the self-stabilizing version, presented in the following section, of the above protocol. A basic knowledge of Petri nets syntax and semantics is assumed; otherwise, a good introduction to this topic can be found in Reisig (1985).

2.2 The correctness proof of the protocol

Petri nets are generic enough to provide modelling of a wide variety of systems. Although due to their asynchronous nature, they are more commonly used for dis-

Figure 1: The cycle of visits for robot r_i

tributed systems. However, their use in distributed systems by no means excludes applications to the field of robotics. In fact, robots themselves form part of the distributed system, and the activity of each one of them can be easily modelled by a local Petri net. Figure 1 shows the local Petri net model associated with an individual robot r_i. We denote it by $N_i = (P_i, T_i, Pre_i, Post_i, M0_i)$ where

1. $P_i = \{p_{(i,0)}, \ldots, p_{(i,j)}, \ldots, p_{(i,n_i-1)}\}$ is the set of places. When $p_{(i,j)}$ is marked, r_i is going to its jth location, or waiting there for its peer.

2. $T_i = \{t_{f(i,0)}, ..., t_{f(i,j)}, ..., t_{f(i,n_i-1)}\}$ is the set of transitions. When $t_{f(i,j)}$ is fired, the communication has happened at the jth location and the robot goes to its next location.

3. Pre_i is the precondition matrix, $Pre_i : P_i \times T_i \rightarrow \{0, 1\}$ defined, according to the behaviour, i.e.

$$Pre_i(p,t) = \begin{cases} 1 & if\ p = p_{(i,j)}\ and\ t = t_{f(i,j)}\ for\ some\ j \\ 0 & otherwise \end{cases}$$

4. $Post_i$ is the postcondition matrix, $Post_i : P_i \times T_i \rightarrow \{0, 1\}$ defined, according to the behaviour, i.e.,

$$Post_i(p,t) = \begin{cases} 1 & if\ p = p_{(i,(j+1)\ modulo\ n_i)}\ and\ t = t_{f(i,j)}\ for\ some\ j \\ 0 & otherwise \end{cases}$$

5. $M0_i$ is the initial marking defined, according to the behaviour, i.e.

$$M0_i(p_{(i,j)}) = \begin{cases} 1 & if\ \ j = 0 \\ 0 & otherwise \end{cases}$$

As a result of the point-to-point communication between robots, a global Petri net is formed by the union of local Petri nets where transitions with the same identity are merged. Thereby, the scheduling protocol is modelled by the global Petri net $\mathcal{N} = (P, T, Pre, Post, M0)$ where

Figure 2: A global Petri net model for an instance of the protocol

1. $P = \biguplus P_i$, is the disjoint union of places of local Petri net models.
2. $T = \bigcup T_i$, is the (non disjoint) union of transitions of local Petri net models.
3. Pre, $Post$ are the Precondition and Postcondition matrices, defined from $P \times T$ over $\{0, 1\}$, by

$$Pre(p,t) = \begin{cases} Pre_i(p,t) & if \;\; p \in P_i \;\; and \;\; t \in T_i \, for \, some \, i \\ 0 & otherwise \end{cases}$$

$$Post(p,t) = \begin{cases} Post_i(p,t) & if \;\; p \in P_i \;\; and \;\; t \in T_i \, for \, some \, i \\ 0 & otherwise \end{cases}$$

4. $M0$ the initial matrix is defined by $M0(p) = M0_i(p)$ if $p \in P_i$.

For instance, consider a system formed by a team of six autonomous robots and six locations. The following table 1 represents the array of locations for each robot. The graphical version of the corresponding global Petri net model of the above system is given by figure 2.

Robots	r_1	r_2	r_3	r_4	r_5	r_6
Locations	1 3 5	2 4 6	1 2	3 4	5	6

Table 1: The locations array

By construction, the global net $\mathcal{N}$ belongs to a particular subclass of Petri nets called *event graphs* defined by the restriction that each place has exactly one input transition and one output transition. In such nets, most of the behavioural properties including *liveness* and *deadlock avoidance* are structurally characterized. A net is said to be live when, whatever the state reached by the net, all the transitions remain fireable in the future. The following lemma of the theory of event graphs will be sufficient for our purposes. We recall the proof since the associated constructions will be used in the proof of self-stabilization.

Lemma 1 *Let $\mathcal{N}$ be an event graph such that every cycle has an initially marked place, then $\mathcal{N}$ is live.*

Proof: An important observation is that given a cycle $\mathcal{C}$ of the event graph $\mathcal{N}$, the only transitions that produce or consume tokens in the places of the cycle are the transitions of the cycle. Thus, the number of tokens in every cycle remains constant.

Suppose that every cycle is initially marked. By the previous observation, we claim that every cycle of every reachable marking M, is also initially marked. For such a marking M, we define a binary relation *helps*$_M$ between transitions such that t *helps*$_M$ t' if and only if there is a place p with $M(p) = 0$ and which is an output place for t and an input one for t'. Let *precedes*$_M$ be the transitive closure of *helps*$_M$. We claim that *precedes*$_M$ is a partial order; otherwise, there may exist transitions t and t' for which t *precedes*$_M$ t' and t' *precedes*$_M$ t. According to the definition of *precedes*$_M$, this implies that there is a path from t to t' and a path from t' to t where every place is unmarked. As usual, one can extract from the concatenation of these two paths, an unmarked cycle contradicting the assumption.

Since every partial order on a finite set can be extended to at least one total order, let $t_1, \ldots, t_n$ be the ordered set of transitions induced by *precedes*$_M$. We claim that $t_1 \ldots t_n$ is a firing sequence starting from the marking M. Indeed, t_1 is fireable since all its input places are marked in M. By induction, the firing of the sequence $t_1 \ldots t_i$, starting from M, leads to a new marking, M'. Thereafter, all the input places of the transition t_{i+1} are marked in M' since either they were already marked in M, or a token has been produced in it by the firing of some t_j with $j \leq i$ and not consumed since a transition does not share its input places. Therefore, the firing sequence can be extended to t_{i+1} and the net $\mathcal{N}$ is live. ■

The correctness of the protocol is expressed in the following proposition.

Proposition 2 *Let $\mathcal{N}$ be a net modelling the protocol for some system, then $\mathcal{N}$ is live.*

Proof: Consider a cycle $\mathcal{C}$ of the event graph $\mathcal{N}$ and let t_k be the transition with the smallest identifier occurring in this cycle, $p_{(i,j)}$ the input place of t_k and $t_{k'}$ the input transition of $p_{(i,j)}$ in the cycle. By construction of $\mathcal{N}$, $k = f(i,j)$ and

$k' = f(i, (j-1)\, modulo\, n_i)$. The choice of t_k implies that $k < k'$, although f is increasing with regard to its second argument. Thus, the only possible value for j is 0. As $p_{(i,0)}$ is initially marked, we have proved that every cycle has an initially marked place. By lemma 1, $\mathcal{N}$ is live. ■

This result can be straightforwardly generalized to the case of n-ary rendez-vous between robots. However, the networks we study are useful due to their flexibility. Introducing n-ary rendez-vous with $n > 2$ decreases such flexibility. So for sake of simplicity, we will restrict ourselves to the initial case of binary synchronization.

3 A self-stabilizing scheduling protocol

A possible enhancement to the previous protocol would be to add fault tolerant properties. In this section, we present a randomized self-stabilizing scheduling protocol based on the previous one. To simplify the discussion and bring forth the fundamental issues, the protocol maintains the following additional assumptions

1. The robots cannot observe the absolute time of their actions, but they have access to timers. A timer is a real-valued variable, whose value continuously decreases in time. More precisely, each robot r_i has a timer, denoted $timeout_i$ and taking any value in the range 0 to $N + 1$, that wakes it up on expiration. Note that these timers run exactly, in the sense that during δ time unit (tu) they decrease by exactly δ . We shall discuss this assumption, in the concluding remarks.

2. The trip between two locations takes at most 1 tu. This hypothesis can always be fulfilled by an appropriate choice of the time unit.

3. Each robot r_i maintains an hardwired array of its locations, sorted by increasing order of the locations identity and denoted $MP_i[0 \ldots n_i - 1]$.

4. The robots are equipped with sensors for detecting their positions. More precisely, each robot r_i has a sensor, giving its current position, denoted $position_i$ and holding any value in the set $\{0, \ldots, n_i - 1\} \cup \{nowhere\}$, indicating either the index, in MP_i, of the location where r_i is waiting, or that r_i is between two locations.

3.1 Description of the protocol

The behaviour of a robot is event-driven: the occurrence of an event triggers the execution of a code depending also on its current state. In our case, there are two events: the timer expiration and the detection of another robot. We denote such an event a peer detection with the obvious meaning that the two robots are both present at some location. We do not consider that the arrival at a location is an event; instead when a robot reaches a location, it just stops. As a robot refills its

timer to 1 tu before going to a new location, the timer will expire after the end of the trip, and then the robot will execute the actions corresponding to the arrival. The crucial point here is that, with this mechanism, *the duration of a trip between two locations becomes exactly* 1 tu. A variable $status_i$, that takes as value either $moving$ or $waiting$, has a special role on the behaviour of the robot w.r.t. the events handling. When this variable is set to $moving$, the robot can neither detect another robot, nor can it be detected by another one. Looking at the proposed protocol, this means that even if a robot arrives at a destination where its peer is already waiting, the communication between them will happen *only after the timer of the arriving robot expires*.

As shown in the program, a robot has four actions: SYNC, WAIT, RECOVER and MISS. SYNC and WAIT correspond to the actions of the original algorithm. In order to recognize that a timer expiration corresponds to an arrival, we use the variable $status_i$. It is set to $moving$ when the robot goes to a new location, and set to $waiting$ when the timer of a robot arriving at a location expires. However, WAIT is different from the corresponding action of the previous algorithm as the robot sets its timer to $N + 1$ (recall that N is the number of the locations). When the timer of a robot arriving at a location expires and a peer is already waiting then it will firstly execute its WAIT action, and as its status is becoming $waiting$ both will execute their SYNC action.

When recovering from a crash, the timer of a robot triggers an action. The action RECOVER is executed by a robot at most once in our protocol (depending on the initial state), and necessarily as the first action of the robot. It happens if the robot is between two locations after the crash. Then the robot goes to its first location.

The key action for the stabilization is MISS. It happens either initially, or when the robot is waiting for a peer at a location and its timer has expired. Then the robot makes a (uniform) random choice between two behaviours:

- it waits again for 1 tu;
- it goes to its first location.

When it is called, the random function Uniform-Choice sets its single parameter to a value among $\{0, 1\}$.

An execution of this algorithm can be seen as an infinite timed sequence $\{t_n, A_n\}_{n\in\mathbb{N}}$, where $\{t_n\}$ is a strictly increasing sequence of times going to infinity and each A_n is the non-empty set of actions that have been triggered at time t_n (at most two actions per robot in the case when it executes WAIT and immediately after SYNC). With this formalization, we can state what is a stabilizing execution.

Definition 3 *An execution $\{t_n, A_n\}_{n\in\mathbb{N}}$ of the protocol is stabilizing if the number of occurrences of* RECOVER *and* MISS *is finite.*

Constant : N, n_i;
$MP_i[0 \ldots n_i - 1]$;

Timer : $timeout_i \in [0 \ldots N + 1]$;

Sensor : $position_i \in \{0, \ldots, n_i - 1\} \cup \{nowhere\}$;

Variables : $status_i \in \{waiting, moving\}$;
$choice_i \in \{0, 1\}$;

ON PEER DETECTION // SYNC
// on r_i arrival or on peer arrival while the other is already waiting
// necessarily $status_i$ is waiting
Exchange messages;
Refill($timeout_i$,1);
$status_i = moving$;
Go to $MP_i[(position_i + 1)\ modulo\ n_i]$;

ON TIMER EXPIRATION
// r_i arrives at the location
If $(position_i != nowhere)$ And $(status_i == moving)$ **Then** // WAIT
Refill($timeout_i$,$N + 1$);
$status_i = waiting$;
Endif

// recovery from a crash while the robot were between two locations
If $(position_i == nowhere)$ **Then** // RECOVER
Refill($timeout_i$,1);
$status_i = moving$;
Go to $MP_i[0]$;
Endif

// expiration of the timer while r_i is waiting for a peer
If $(position_i != nowhere)$ And $(status_i == waiting)$ **Then** // MISS
Uniform-Choice($choice_i$);
Case ($choice_i$)
0 : Refill($timeout_i$,1);
1 : Refill($timeout_i$,1);
$status_i = moving$;
Go to $MP_i[0]$;
Endcase
Endif

Figure 3: Protocol specification for robot r_i

In other words, after a finite time, the protocol behaves like the original algorithm. Let us remind that RECOVER can occur at most once per robot. The next section will be devoted to prove the following proposition.

Proposition 4 *Given any initial state, the probability that an execution will stabilize is 1 and the mean time until the stabilization is finite.*

4 Proof of stabilization

Without loss of generality, we consider that the initial state is a state obtained after each robot has executed at least one action. Thus we do not have to take into account the action RECOVER. With this hypothesis and for a better understanding of the protocol, a state graph of a robot is presented in figure 4.

4.1 Probabilistic semantics of the protocol

We assume that the code execution is instantaneous: indeed, the time that takes for a robot to execute internal computation is negligibly small with regard to the time it takes for the robot to move to a new location. Thus in our protocol, the single source of indeterminism is the random choice of the MISS action since all trips take exactly 1 tu. Consequently, the probabilistic semantics of our protocol is a Markov chain whose description is given below.

A state of this Markov chain is composed by the specification of a state for each robot. The state of a robot r_i is defined by its vector $< s_i, l_i, to_i, \alpha_i >$, where:

- s_i: the robot's status that takes value in the set $\{waiting, moving\}$ depending on whether the robot is waiting at a location or moving to it,
- l_i: the location where the robot is waiting or moving to,
- to_i: given by the formula $\lceil timeout_i - 1 \rceil$ where $\lceil x \rceil$ denotes the least integer greater than, or or equal to x. to_i takes its value in $\{0, \ldots, N\}$,
- α_i: given by the formula $\alpha_i = timeout_i - to_i$, it takes its value in $]0 \ldots 1]$. We will call it the *residual value*.

The last attributes deserve some attention. As we consider states after the execution of the actions, the variables $timeout_i$ are never null: this explains the range of these attributes. Moreover, these attributes are simply a decomposition of $timeout_i$. However the interest of this decomposition will become clear in the next paragraph. So, a state e will be defined by: $e = \prod_{i=1}^{m} < s_i, l_i, to_i, \alpha_i >$.

Let us note that the set of states is infinite and even uncountable since the α_i's take their values in an interval of $\mathbb{R}$. However, we show that we can lump this chain into a finite Markov chain with the help of an equivalence relation that fulfills the conditions of strong lumpability Kemeny and Snell (1960).

Figure 4: A state graph for r_i

Definition 5 *Two states* $e^1 = \prod_{i=1}^{m} < s_i^1, l_i^1, to_i^1, \alpha_i^1 >$ *and* $e^2 = \prod_{i=1}^{m} < s_i^2, l_i^2, to_i^2, \alpha_i^2 >$ *are equivalent if:*

1. $\forall\, i, s_i^1 = s_i^2, l_i^1 = l_i^2, to_i^1 = to_i^2,$
2. $\forall\, i, j,\ \alpha_i^1 < \alpha_j^1 \Longleftrightarrow \alpha_i^2 < \alpha_j^2$

An equivalence class (denoted by c) of this relation is characterized by: $c = \prod_{i=1}^{m} < s_i, l_i, to_i > \times\, position$, where $position$ represents the relative positions of the α_i's. It is easy to show that there are at most $m! \cdot 2^{m-1}$ distinct positions. Thus, the number of equivalence classes is finite. The next proposition establishes the condition of strong lumpability.

Proposition 6 *Let* c *and* c' *two equivalence classes,let* e^1 *and* e^2 *be two states of the class* c*, then:*

$$\sum_{e \in c'} P[e^1, e] = \sum_{e \in c'} P[e^2, e]$$

where P *denotes the transition matrix of the Markov chain.*

Proof: The proof is omitted, see El Haddad and Haddad (2003). ■

A Markov chain can be viewed as a graph where there is an edge between one state s and another s' iff there is a non null probability to go from the former to the latter (i.e. $P[s, s'] \neq 0$). The edge is labelled by this probability. The following lemma (only valid for finite chains) will make the proof of correctness easier.

Lemma 7 (Feller 1968) *Let* S' *be a subset of states of a finite Markov chain. Let us suppose that for any state* s*, there is a path from* s *to some* $s' \in S'$*. Then whatever the initial state, the probability to reach (some state of)* S' *is 1 and the mean time to reach it is finite.*

Figure 5: The level of transitions of a deadlock-free marking

4.2 Stable states

In this subsection, we exhibit a condition on states that ensures that, in an execution starting from a state fulfilling such a condition, the MISS action will never occur. We need some preliminary definitions based on the Petri net modelling of the original protocol.

Definition 8 *Let* $e = \prod_{i=1}^{m} < s_i, l_i, to_i, \alpha_i >$ *be a state of the system. Then the marking M(e) of the net* N *modelling the protocol is defined by:* $M(e)(p_{i,j}) = 1$ *If* $l_i = f(i,j)$ *Else* 0*.*

In fact, the marking $M(e)$ is an abstraction of the state e where the timed informations and the status of the robot are forgotten.

Definition 9 *Let* N *be a net modelling the protocol and* M *be a marking of* N*, then* M *is said to be deadlock-free if for the marking* M*, all the cycles of* N *are marked.*

In a state modelled by a deadlock-free marking, if we execute the original protocol, then no deadlock will never happen. However, due to the values of the timer, it may happen that for a state e with $M(e)$ being deadlock-free, a MISS action happens (for instance, on timer expiration of a waiting robot while its peer is still moving). Thus we must add timed constraints to the state e in order to forbid such behaviour.

If M is deadlock-free, then the relation $helps_M$ introduced in lemma 1 defines a directed acyclic graph (DAG) between transitions. We define $level_M(t)$ as the length of the longest path of this DAG ending in t. Here the length of a path is the number of vertices of this path. In figure 5, we have represented the level of transitions for the initial marking of the net of figure 2. We are now ready to define our condition on states.

Definition 10 *Let* $e = \prod_{i=1}^{m} < s_i, l_i, to_i, \alpha_i >$ *be a state of the system. Then* e *is stable if M(e) is deadlock-free and,* $\forall i,\ s_i = waiting \Rightarrow to_i \geq level_{M(e)}(t_{l_i})$*.*

The next lemma shows that the definition of stable states is appropriate.

Lemma 11 *In an execution starting from a stable state, the action* MISS *will never happen.*

Proof: We will proceed by induction on the states of the system at discrete time $0, 1, 2, \ldots$. We note e^n the state of the system at time n. e^0 is the initial stable state. Be aware that these states do not correspond to the successive states of the Markov chain, but it does not matter since we will not use here any probabilistic argument. Our induction hypothesis is that until time n no MISS action has happened, and e^n is stable. For $n = 0$, it is just the hypothesis of the lemma. Let us examine what happens between time n and $n + 1$.

Let us look at a robot waiting at a location at time n. Since its timer is greater than 1 (by the stability hypothesis and the fact that $\alpha_i > 0$), it will not expire until $n+1$. Let us now look at a robot moving to a location at time n. It will arrive during the interval $[n \ldots n+1]$ and will refill its timer to $N + 1$. In both cases, either a SYNC will happen and the robot will be moving at time $n + 1$, or it will still be waiting. Thus we have proved that no MISS action happens during this interval.

It remains to show that e^{n+1} is a stable state. Since no MISS action happens during the interval, the execution (without taking into account the timer values) corresponds to the execution of the non self-stabilizing protocol. Thus $M(e^{n+1})$ is a marking reached by a firing sequence from $M(e^n)$, and so all the cycles are marked in this new marking.

Let t be a transition of level 1 for $M(e^n)$. t has its two places marked, meaning that the two associated robots are either waiting at the corresponding location, or moving to it. Thus the synchronization will happen before time $n + 1$.

Let t be a transition of level > 1 for $M(e^n)$. t has one of its places unmarked, that means that one of the robots associated with the corresponding location is neither waiting at this location, nor it is moving to. Thus a synchronization at the location is impossible during this interval. So t will not be fired during the interval.

Suppose now that a robot r_i is waiting at time $n + 1$ at a location. If this robot has arrived during the interval, it has set its timer to $N + 1$, and thus at time $n + 1$, to_i is still equal to N, which is an upper bound for the level.

Finally, suppose that this robot has been waiting during the whole interval. Then its timer (and so to_i) is decreased by one at time $n + 1$, but the level of the corresponding transition was greater than one at time n and has not been fired. All the transitions of level 1 have been fired, so its level at time $n + 1$ is also decreased by 1 (since the paths to this transition in the new DAG are exactly the paths to it in the old DAG truncated by their origin). Thus the timed constraints are still verified and e^{n+1} is a stable state. ■

4.3 From an initial state to a stable state

In this section, we show that given any initial state, there is a path from this state to a stable state in the Markov chain. Thus the proposition 4 will follow almost

directly from lemmas 7 and 11. The single non trivial observation to make is that given two equivalent states s and s' (see definition 5), then s is stable iff s' is stable since the stability does not involve the residual times. Thus the path found below gives a path in the finite aggregated Markov chain where the final state is a set of stable states.

Lemma 12 *Given any initial state, there is a path in the Markov chain from this state to a stable state.*

Proof: As we look for a path in the Markov chain, each time the MISS action happens, we can choose its random output. So, when in what follows we will choose, during a part of execution, the first choice (staying at the location), we will say that we simulate the original algorithm.

If the initial state is stable, then we are done. So we suppose that the initial state e is not stable. We examine the two following cases:

1. $M(e)$ is deadlock-free.

   Here the timed constraints of stability are not verified by e. By simulating the original algorithm, we claim that a stable state will be reached. First, all successive markings associated with the states will be deadlock-free since they are reachable from $M(e)$ in the net N.

   Second, we decompose time into intervals of 1 tu. During each interval, all the locations corresponding to the transitions of level 1 will be the support of SYNC actions. A robot that will execute such a SYNC action will have its timed constraint fulfilled since it is moving. Moreover, using exactly the same proof as the one of lemma 11, it can be shown that when a timed constraint is fulfilled, it will always be fulfilled. Thus after each robot has executed at least one SYNC action, we have reached a stable state.

2. $M(e)$ is not deadlock-free.

   Since there is a chain of synchronization locations between any pair of robots, applying the original algorithm would lead us to a global deadlock. Thus we simulate the original algorithm until every robot is blocked alone at a location, and then has executed at least once its MISS action. This means that all timers have their values ≤ 1.

   Now we choose for every robot the second alternative of the MISS action. All these actions happen in less than 1 tu. So after the last MISS action has been executed, every robot is still moving to its first location. In this state denoted by e', $M(e')$ is the initial marking of the net modelling the original protocol. Thus $M(e')$ is deadlock-free and we complete the current path by the path of the first case. ■

5 A routing table maintenance algorithm

In this section, the robots are no more anonymous since they exchange messages using the identities to refer the destination. A natural method for trying to provide routing in multi-robots networks is to consider each mobile robot as a router and to run a routing protocol between them. The objective of most routing protocols is to find a path to the destination which is optimal with respect to a given metric. In this section, we present a minimum-hop, self-stabilizing algorithm for maintaining routing tables which is an extension of Dolev *et al.* (1989). Whereas routing via minimum-hop is still required, the routing decision made when forwarding a message *depends* on the destination of the message (and the contents of the routing tables) and the current position of the forwarding robot.

Since the management of each destination robot is independent, we fix an arbitrary robot r_0 called the destination robot.

The algorithm is based on a communication *oriented* graph where the vertices of the graph are the locations of the robots with two vertices per location (i.e. one per peer). There is an edge between u and v iff:

- $\exists\, r, r'$ two robots $\exists\, l$ a common location of r and r' such that $u = (r, l)$ and $v = (r', l)$, or
- $\exists\, r_i$ a robot $\exists\, 1 \leq j \leq n_i$ such that $u = (r_i, f(i, j))$ and $v = (r_i, f(i, (j + 1) \bmod n_i))$

Let us suppose that upon stabilization, the algorithm computes a forest of n_0 trees each one rooted at a location of r_0 spanning the graph with minimum-hop paths towards r_0. Let us consider a robot r currently at a location l shared with r' which needs to send a message to r_0. If the father of (r, l) is the next location of r then the robot keeps the message until it reaches the next location. Else if the father is (r', l) then the robot sends the message to r'. With such a data srtcuture, it is straightforward to show that the message will reach r_0.

The self-stabilizing algorithm we propose is depicted in figure 6. Each robot r_i maintains the following variables:

- $distance_i[0 \ldots n_i - 1]$: an array of integer, where $distance_i[j]$ is the estimation by r_i of the distance between its location $MP_i[j]$ and the destination r_0. A distance variable can store an integer value between 0 and $2N - 1$, where N is the number of locations in the whole system.
- $exit_i[0 \ldots n_i - 1]$: an array of booleans. If $exit_i[j]$ is true then the father of the vertex $(r_i, MP_i[j])$ is the vertex $(r_k, MP_i[j])$ with r_k corresponding to the peer of r_i for the location $MP_i[j]$. If $exit_i[j]$ is false then the father of the vertex $(r_i, MP_i[j])$ is the vertex $(r_i, MP_i[j + 1\, mod\, n_i])$

Variables : $distance_i[1 \ldots n_i] \in [0 \ldots 2N-1]$;
$exit_i[1 \ldots n_i]$;

For node r_0 only
do periodically
for $j = 0$ **to** $n_0 - 1$ **do** $distance_0[j] = 0$;
enddo

On synchronization with r_k at location $MP_i[j]$
Send a $distance_i[j]$ to r';
Receive d from r';
if $(r_i! = r_0)$ **then**
$d' = min(d,\ distance_i[(j+1) \bmod n_i])$;
if $(d' < 2N-1)$ **then**
$distance_i[j] = d' + 1$;
$exit_i[j] = (d == d')$;
endif
endif

Figure 6: The routing table maintenance algorithm

The exemple given in table 2 specifies the part of the routing table of r_1 w.r.t. the destination r_6, in the system illustrated by the figure 2. For instance, if r_1 standing at location 5 wants to send a message to r_6, it will wait to be at location 1 and will forward it to r_3.

	distance	exit
$(r_1, 1)$	6	$true$
$(r_1, 3)$	5	$true$
$(r_1, 5)$	7	$false$

Table 2: The stabilized routing table entries of r_1 towards r_6

The $distance$ table of r_0 is periodically set to 0. Thus starting from any state this table is correct after one itération of this loop. An entry of the tables of any other robot corresponding to location l is updated when synchronizing at this location by:

- The exchange of the current estimation of the distance by the robots between this location and r_0.

- The new estimation is the minimum between the distance of the peer incremented by one and the distance of the next location of the robot incremented by one. The corresponding entry of the $exit$ table is set to true iff the minimum is obtained by the distance of the peer.

6 Proof of correctness

Since this algorithm is composed with the previous one, we suppose that in an initial state, the scheduling algorithm has stabilized. Let us introduce some concepts associated to an execution. An execution sequence is decomposed into successive rounds. Each round starts at the end of the previous one and finishes when all the robots have performed at least one visit per location. By convention, the first round starts when r_0 has set to zero its table $distance$.

A *floating distance* in some configuration c is a value in a distance variable that is smaller than the real distance. The *smallest floating distance* in some configuration c is the smallest value among the floating distances. If there is no floating values in some configuration, then $sfd = \infty$. The lemma below, shows that, in every execution, a safe configuration is reached.

Lemma 13 *For every $k \geq 0$ and for every configuration that follows the first k rounds, it holds that:*

Assertion 1: If there exists a floating distance, then the value of sfd is at least k

Assertion 2: The nodes with distance values less than or equal to k consist a forest of minimum-hop trees.

Preuve : Let sfv^k denote the smallest floating distance at the end of the kth round and sfd^0 the initial smallest floating distance. We prove the lemma by induction over k.

For $k = 1$, the distances stored in the variables are non-negative; thus the value of the smallest floating distance is at least 0 in the first configuration. This proves assertion 1. To prove assertion 2, note that before the first round, the destination robot r_0 sets all its distances variables to 0. Once these distances are computed, they will never be changed. Therefore, each location of r_0 is the root of a tree and the assertion 2 holds as well.

Assume correctness for $k \geq 0$ and prove for $k + 1$. Let $m \geq k$ be the smallest floating distance in the configuration that follows the first k rounds. During the round $k + 1$, each robot that recompute its distance variables, either assigns the correct value or chooses m as the smallest value and assigns $m + 1$ to its distance variable. Therefore, the smallest floating distance value is $m + 1$. This proves assertion 1.

Since the smallest floating distance is $m \geq k$, it is clear that each robot that receives the distance k of a peer, computes the smaller distance value between k and the distance of its next location. The new value is either $k + 1$ or the distance of its next location incremented by one. In the case of $k + 1$, the robot distance value equals its real distance. In the other case, it equals a floating distance greater than or equal to k. Note that, once the value in the variable of robot is equal to its distance from the root, then it joins one of the trees of the forest by choosing either its peer or its next location, as its father in the tree. ■

The next corollary is implied by lemma 13.

Corollary 14 *The routing algorithm presented above stabilizes after* $2N$ *rounds.*

7 Concluding Remarks

We have designed a uniform self-stabilizing scheduling protocol for a network of robots. The interest of this work is twofold. On the one hand, self-stabilization is an important and desirable feature of protocols for these environments. On the other hand, the use of formal models for proofs of stabilizing algorithms is not so frequent. Here, with the help of Petri nets theory, we have simplified the proof of the non stabilizing version of the algorithm. A part of the proof of stabilization is also based on this model.

The assumption that timers run exactly is only important during the stabilization step. Once the algorithm reaches a stable state, we can show that the protocol still works if the timers are prone to small deviations. Moreover in practice, if the stabilization step is not too long, then the deviations of the timers will not disturb it.

The next stage of our research is to prove that no deterministic self-stabilizing algorithm exists for the scheduling task, to explore how should new robots be incorporate in the system without bringing it down and to develop self-stabilizing algorithms for dynamic systems supporting that some robots may join or leave the system.

References

BRACKA, P., MIDONNET, S., and ROUSSEL, G., 2003, Scheduling and Routing in an Ad Hoc Network of Robots. IASTED International Conference on Computer Science and Technology, Cancun, Mexico.

CAO, Y. U., FUKUNAGA, A., and KAHNG, A., 1997, Cooperative mobile robotics : Antecedents and directions. *Autonomous Robots*, 4, 7-23.

DEFAGO, X., and KONAGAYA, A., 2002, Circle formation for oblivious anonymous mobile robots with no common sens of orientation. Workshop on self stabilizing, POMC'02, pp. 97-104.

DIJKSTRA, E., 1974, Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17, 643-644.

DOLEV, S., ISRAELI, A., and MORAN, S., 1989, Self-stabilization of dynamic systems. Proceedings of the MCC Workshop on Self-stabilizing systems, MCC Technical Report No. STP-379-89

DOLEV, S., 2000, *Self-stabilization* (MIT Press).

DOLEV, S., and HERMAN, T., 1999, Parallel composition of stabilizing algorithms. Workshop on Self Stabilizing, country, pp. 25-32.

El HADDAD, J., and HADDAD, S., 2003, Self-stabilizing scheduling algorithm for cooperating robots. ACS/IEEE International Conference on Computer Systems and Applications,Tunisia.

FELLER, W., 1968, *An introduction to probability theory and its applications*, (John Wiley and Sons), volume 1.

HU, H., KELLY, I., KEATING, D., and VINAGRE, D., 1998, Coodination of multiple mobile robots via communication. Proceedings of SPIE, Mobile Robots XIII and Intelligent Transportation Systems, Boston, Massachusetts, pp. 94-103.

KEMENY, J., and SNELL, J., 1960, *Finite Markov Chains* (Van Nostrand, NJ : Princeton).

LUZEAUX, D., 2000, Autonomous small robots for military applications. Conference on Unnamed Groud Vehicle Technology, USA.

PARK, V., and CORSON, M., 1997, A highly adaptive distributed routing algorithm for mobile wireless networks. *Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 3, 1405-1413.

PRENCIPE, G., 2001, Corda : Distributed coordination of a set of autonomous mobile robots. European Research Seminar on Advances in Distributed Systems, Ersads.

REISIG, W., 1985, *Petri Nets: an Introduction* (Springer Verlag).

SCHNEIDER, M., 1993. Self-stabilization. *ACM Symposium Computing Surveys*, 25, 45-67.