Self-Stabilization with r-operators revisited

Sylvie Delaët[†] Bertrand Ducourthial[*] Sébastien Tixeuil[‡]

[†]LRI – CNRS UMR 8623, Université Paris Sud, France
[*]Heudiasyc – UMR CNRS 6599, UTC, Compiègne, France
[‡]LRI – CNRS & INRIA Grand Large, Université Paris Sud, France

Abstract

We present a generic distributed algorithm for solving silents tasks such as shortest path calculus, depth-first-search tree construction, best reliable transmitters, in directed networks where communication may be only unidirectional. Our solution is written for the asynchronous message passing communication model, and tolerates multiple kinds of failures (transient and intermittent).

First, our algorithm is self-stabilizing, so that it recovers correct behavior after finite time starting from an arbitrary global state caused by a transient fault. Second, it tolerates fair message loss, finite message duplication, and arbitrary message reordering, during both the stabilizing phase and the stabilized phase. This second property is most interesting since, in the context of unidirectional networks, there exist no self-stabilizing reliable data-link protocol. The correctness proof subsumes previous proofs for solutions in the simpler reliable shared memory communication model.

1 Introduction

Historically, research in self-stabilization over general networks has mostly covered undirected networks where bidirectional communication is feasible and carried out using shared registers (see [6]). This model permits algorithm designers to write elegant algorithms and proofs. To actually implement such self-stabilizing algorithms in real systems, where processors communicate by exchanging messages, transformers that preserve the self-stabilizing property of the original algorithm are needed. Such transformers are presented in [2, 6], and are based on variants of the alternating bit protocol or the sliding window protocol. A common drawback to these transformers is that they require the receiver of a message to be able to send acknowledgments to the emitter periodically, so that the underlying message passing network must be bidirectional for the transformer to be correct.

Hence, in directed networks, acknowledgment-based transformers cannot be used to run self-stabilizing algorithms in message passing networks, since it is possible that there exist two neighbors in the network that are only connected through a unidirectional link. Moreover, in directed message passing networks, it is generally easy to maintain the set of input neighbors (by checking who has ”recently” sent a message), but it is very difficult (if not impossible) to maintain the set of output neighbors. For instance, in a satellite or a sensor network, a transmitter is generally not aware of who is listening to the information it communicates. Note also that wireless networks can be directed message passing networks, especially when power of emissions are not uniform: a node i can receive a message from j while the converse is not possible.

So, self-stabilizing algorithms that use implicit neighborhood knowledge to compare one node state with those of its neighbors and to check for consistency – a large subset of self-stabilizing algorithms – cannot be used in directed networks.

The particular system hypothesis and the lack of transformers has led authors to design specific self-stabilizing algorithms for directed networks [1, 4, 10, 5, 11, 8].

The solutions [1, 4, 5, 8] are "classical" in the sens that a self-stabilizing layer (or mechanism) is added to a well known (non-stabilizing) protocol to ensure stabilization. This typically induces a potential overhead (extra knowledge, variables, processing are needed). In contrast, [10, 11] are *condition based*: either the algorithm satisfies the condition (and is then stabilizing) or not (and is not stabilizing). So, no overhead is induced by adding the self-stabilizing property to the original algorithm (the original algorithm is not changed). The two solutions of [10, 11] are generic (they can solve multiple problem instances with a single parameterized algorithm), but perform in the unidirectional shared memory model. In [11], the atomicity of communication is composite: in one atomic step, a processor is able to read the actual state of all of its neighbors and update its state, while in [10], the atomicity is read-write: in one atomic step, a processor is able to read the state of one neighbor, or update its state, but not both. Both approaches cannot be transformed to perform in unidirectional message passing networks using known self-stabilizing transformers (see above). The two solutions of [4, 5, 8] are specific (a single problem is addressed, the routing problem in [4], the census problem in [5], and the group communication problem in [8]), but perform in directed message passing networks. While [4, 8] assume reliable communications (links do not lose, duplicate or reorder messages), [5] tolerates message loss, duplication, and reordering. [1] proposes a generic solution in the message passing model, but assumes that communications are reliable (with FIFO links), that nodes have unique identifiers, and that the network is strongly connected, three hypothesis that we do not make.

Our Contribution. In this paper, we concentrate on providing a generic algorithm (that can be instantiated to solve silent tasks, see [7]), that performs on general directed message passing networks. Our solution is not only self-stabilizing (it recovers in finite time from any initial global state), it also tolerates fair message loss, finite duplication, and arbitrary reordering both in the stabilizing and in the stabilized phase. Nice properties of our approach are that the network need not be strongly connected, and nodes need not know whether the network contains cycles, and no upper bound on the network size, diameter, or maximum degree. However, if such information is known, the stabilization time can be significantly reduced.

We provide, in more details, a parameterized algorithm that can be instantiated with a local function. Our parameterized algorithm enables a set of silent tasks to be solved self-stabilizingly, provided that these tasks can be expressed through local calculus operations

Reference	Overhead	Atomicity	Reliability	Algorithm
[1]	yes	send/receive atomicity	reliable	generic (total order)
[4]	yes	send/receive atomicity	reliable	specific (routing)
[5]	yes	send/receive atomicity	unreliable	specific (census)
[8]	yes	send/receive atomicity	reliable	specific (group communication)
[11]	no	composite atomicity	reliable	generic (partial order on $\mathbb{S}$)
[10]	no	read/write atomicity	reliable	generic (total order on $\mathbb{S}$)
This paper	no	send/receive atomicity	unreliable	generic (total order on $\mathbb{S}$)

Figure 1: A summary of related self-stabilizing algorithms in directed networks.

called r-operators that operate over a set $\mathbb{S}$. The r-operators are general enough to permit applications such as shortest path calculus and depth-first-search tree construction on arbitrary graphs while remaining self-stabilizing.

The main differences between this paper and the most closely related work [10] are twofold. First, we consider an unreliable message passing communication network, instead of a reliable shared memory system. As noted above, unidirectional read-write systems cannot be emulated in message passing networks by means of a known self-stabilizing transformer. The key difference is that shared registers may hold only the latest written value, while the communications links we consider may hold an unbounded number of (possibly erroneous) messages that can appear again once the network appears to have stabilized (due to the reordering assumption). Second, the proof technique that we use here is based on a completely different idea than that of [10]. In [10], it is first proved that a terminal configuration is eventually reached starting from any initial configuration, and then (using a complicated induction argument) that this terminal configuration is in fact legitimate. In contrast, in message passing networks, self-stabilizing systems cannot be terminating (otherwise deadlock situations could occur, see [14]), so the proof argument here is to prove the following two invariants: *(i)* the state of each processor is eventually lower than (or equal to) its legitimate state (in the sense of the order defined on $\mathbb{S}$), and *(ii)* the state of each processor is eventually greater than (or equal to) its legitimate state, so that the state of each processor is eventually legitimate. Not only is this new proof simpler and more elegant than that of [10], it also permits algorithm designers to abstract the communication media that is used, so that the same proof applies for shared memory and unreliable message passing systems.

In Figure 1, we capture the key differences between our protocol and the aforementioned related solutions ([1, 4, 10, 5, 11]) in general directed networks regarding the following criteria: communication, overhead, atomicity, reliability, and algorithm nature.

Outline. Section 2 presents a model for distributed systems we consider. Section 3 describes our self-stabilizing parameterized algorithm for general directed networks, along with our system hypothesis. Our main result is presented, and is illustrated by an example. The sketch of the proof of correctness is also given. Section 4 details the proof. An interesting point is that this proof subsumes previous proofs for solutions in the simpler reliabl shared memory model. In Section 5 we show how the very algebraic nature of our scheme makes it suitable for *ad hoc* and sensor wireless networks, considering the unreliable communication mechanisms that are provided in those networks. Concluding remarks are proposed in

Section 6.

2 Model

Processors and links. Processors use *unidirectional communication links* to transmit messages from an origin processor o to a destination processor d. The link is interacting with one input port of d and one output port of o. A link may hold an arbitrary number of messages (although our algorithm also works for bounded capacity links). Depending upon the way messages are handled by a communication link, several properties can be defined on a link. A complete formalization of these properties is proposed in [16]. We only enumerate those that are related to our algorithm. There is a *fair loss* when, infinitely many messages being emitted by o, infinitely many messages are received by d. There is *finite duplication* when every message emitted by o may be received by d a finite (yet unbounded) number of times. There is *reordering* when messages emitted by o may be received by d in a different order than that they were emitted. There is *eventual delivery* if any message that is not lost is eventually received (*i.e.* no message remains forever in a communication link).

Distributed system. A *distributed system* is a 2-tuple $\mathcal{S} = (\mathcal{P}, \mathcal{L})$ where $\mathcal{P}$ is the set of processors and $\mathcal{L}$ is the set of communication links. Such a system is modeled by a *directed graph* (also called *digraph*) $G = (V, E)$, defined by a set of vertices V and a set E of edges (v_1, v_2), which are ordered pairs of vertices of V $(v_1, v_2 \in V)$. Each vertex u in V represents a processor P_u of system $\mathcal{S}$. Each edge (u, v) in E represents a communication link from P_u to P_v in $\mathcal{S}$. In the remainder of the paper, we use interchangeably processors, nodes, and vertices to denote processors, and links and edges to denote communication links. Also, we use the standard notation $A \setminus B$ to denote the set of elements that are in set A but nor in set B.

Graph notations. The *in-degree* of a vertex v of G, denoted by δv is equal to the number of vertices u such that the edge (u, v) is in E. The incoming edges of each vertex v of G are indexed from 1 to δv. A *directed path* P_{v_0,v_k} in a digraph $G(V, E)$ is an ordered list of vertices $v_0, v_1, \ldots, v_k \in V$ such that, for any $i \in \{0, \ldots, k-1\}$, (v_i, v_{i+1}) is an edge of E (*i.e.*, $(v_i, v_{i+1}) \in E$). The *length* of this path is k. If each v_i is unique in the path, the path is *elementary*. The set of all elementary paths from a vertex u to another vertex v is denoted by $\mathcal{X}_{u,v}$. A *cycle* is a directed path P_{v_0,v_k} where $v_0 = v_k$. The *distance* between two vertices u, v of a digraph G (denoted by $\mathrm{d}_G(u, v)$, or by $\mathrm{d}(u, v)$ when G is not ambiguous) is the minimum of the lengths of all directed paths from u to v (assuming there exists at least one such path). The *diameter* of a digraph G is the maximum of the distances between all couples of vertices in G between which a distance is defined. Finally, we denote as Γ_v^- (resp. Γ_v^+) the set of predecessors (resp. successors) of a vertex $v \in V$, that is the set of all vertices $u \in V$ such that there exists a path starting at u (resp. v) and ending at v (resp. u). The predecessors (resp. successors) u of v verifying $\mathrm{d}_G(u, v) = 1$ (resp. $\mathrm{d}_G(v, u) = 1$)) are called *direct-predecessors* (resp. *direct-successors*) and their set is denoted Γ_v^{-1} (resp. Γ_v^{+1}).

Configurations and executions. The global system state, called a *system configuration* (or simply *configuration*) and generally denoted c, is the union of (i) the states of memories of processors of $\mathcal{P}$ and (ii) the contents of communication links of $\mathcal{L}$. The set of configurations is denoted by $\mathcal{C}$. The part of information in a configuration $c \in \mathcal{C}$ related to the processors of $\mathcal{P}$ is denoted by $c_{|\mathcal{P}}$; the part related to a given processor $P \in \mathcal{P}$ is denoted by $c_{|P}$.

Starting from an *initial configuration* c_1, an *execution* $e_{c_1} = c_1, a_1, c_2, a_2, \ldots$ is a maximal alternating sequence of configurations and actions of such that, for any positive integer i, the transition from configuration c_i to configuration c_{i+1} is done through execution of action a_i. Maximal means that either the computation is infinite, or the computation is finite and no action is enabled in the final configuration. The notations $\mathcal{E}_c$, $\mathcal{E}_C$ and $\mathcal{E}$ denote respectively the set of all executions starting (i) from the initial configuration c, (ii) from any configuration $c \in C \subset \mathcal{C}$, or (iii) from any configuration of $\mathcal{C}$ ($\mathcal{E}_{\mathcal{C}} = \mathcal{E}$). The ordered list $c_1, c_2, \ldots \in \mathcal{C}$ of the configurations of an execution $e = c_1, a_1, c_2, a_2 \ldots$ is denoted by $e_{|\mathcal{C}}$. In the rest of this paper, we adopt the following convention: if $c_i \in e_{|\mathcal{C}}$ appears before $c_j \in e_{|\mathcal{C}}$, then $i < j$.

Distributed algorithms resolve either static tasks (*e.g.*, distance computation) or dynamic tasks (*e.g.*, token circulation). The aim of static tasks is to compute a global result, which means that after a running time, processors always produce the same output (*e.g.*, the distance from a source). A static task is characterized by a final processor output o_P for any processor $P \in \mathcal{P}$, called *legitimate output*. A *legitimate configuration* c for this task satisfies $c_{|P} = o_P$ for any processor $P \in \mathcal{P}$. A distributed protocol designed for solving a given static task is correct if the distributed system $\mathcal{S}$ running this protocol reaches in finite time a legitimate configuration for this task.

Self-stabilization. A set of configurations $C \subset \mathcal{C}$ is *closed* if, for any $c \in C$, any possible execution $e_c \in \mathcal{E}_c$ of system $\mathcal{S}$ whose c is initial configuration only contains configurations in C. A set of configurations $C_2 \subset \mathcal{C}$ is an *attractor* for a set of configurations $C_1 \subset \mathcal{C}$ if, any execution $e_c \in \mathcal{E}_{C_1}$ contains a configuration of C_2. Let $C \subset \mathcal{C}$ be a non-empty set of configurations. A distributed system $\mathcal{S}$ is *C-stabilizing* if and only if C is a closed attractor for $\mathcal{C}$: any execution e of $\mathcal{E}$ contains a configuration c of C, and any further configurations in e reached after c remains in C. Finally, consider a static task for the distributed system $\mathcal{S}$, and let $L \subset \mathcal{C}$ be the set of the legitimate configurations of $\mathcal{S}$. A distributed protocol designed for solving this static task is *self-stabilizing* if the distributed system $\mathcal{S}$ running this protocol is L-stabilizing.

3 Parametric message passing $\mathcal{PA}$-MP algorithm

In this section, we first describe the distributed system we consider before defining the $\mathcal{PA}$-MP parametrized algorithm. We then introduce the r-operators, that are used as parameters. These operators are derivated from the associative, commutative and idempotent operators (such as the minimum on the integers).

3.1 System

Let $\mathcal{S} = (\mathcal{P}, \mathcal{L})$ be the distributed system we consider in the following. The associated graph composed of processors of $\mathcal{P}$ and communications links of $\mathcal{L}$ is fixed, directed and unknown to the processors of $\mathcal{P}$. Communications between processors are performed by message passing (directed message passing network).

Each processor v of $\mathcal{P}$ is endowed with a local real-time clock mechanism. However, those clocks are use for the sole purpose of being able to perform action based on some timeout mechanism, so our clocks are neither synchronized nor have bounded drift. Each processor v of $\mathcal{P}$ owns an incoming memory denoted as $\texttt{IN}_v$, which is supposed to be unalterable; this can be implemented by a ROM memory (*e.g.*, EPROM), or a memory that is regularly reloaded by any external process (human interface, captor, other independent algorithm, *etc.*). The value of this memory (that will never change) is called *initialization value*. For most provided applications, this initialization value is equal to the identity element of the set $\mathbb{S}$ (except for a limited set of predecessors, see below). Moreover, for each link, starting at processor $u \in \mathcal{P}$ and ending at processor v, there exists a corresponding incoming memory $\texttt{IN}_v^u$ in v, which is used by v to store incoming messages sent by u. Note that $\texttt{IN}_v^u$ contains only one message. A processor v only stores the latest received message from u. In addition, processor v owns an output memory denoted by $\texttt{OUT}_v$. All these memories are private, and can only be read or written by v (note that v only reads $\texttt{IN}_v$, and only writes $\texttt{OUT}_v$). In the following, we identify the name of a memory with the value it contains. In the same way, a message is considered as equivalent to its value.

Processor v performs a calculation by applying an operator $\triangleleft$ (see § 3.3) on all of its incoming memories, and stores the result in its output memory $\texttt{OUT}_v$.

3.2 Algorithm

In this paper, we design a parameterized distributed protocol for Message Passing systems (denoted as $\mathcal{PA}$-MP). This protocol is composed of one local parameterized algorithm per processor v of $\mathcal{P}$, denoted by $\mathcal{PA}$-MP$|_{\triangleleft_v}$, where $\triangleleft_v$ is an operator used as a parameter (parameters could be slightly different on each processor, see Hypothesis 2).

This local algorithm calls three helper functions: $\texttt{Store}_v(m, u)$ stores in the local register $\texttt{IN}_v^u$ the contents of the message m; $\texttt{Evaluate}_v(\triangleleft_v)$ stores in the local register $\texttt{OUT}_v$ the result of the local computation $\triangleleft_v(\texttt{IN}_v, \texttt{IN}_v^{u_1}, \ldots, \texttt{IN}_v^{u_k})$ where $u_1, \ldots, u_k$ are direct predecessors of v ($\in \Gamma_v^{-1}$); $\texttt{Forward}_v$ sends $\texttt{OUT}_v$ to w for each processor $w \in \Gamma_v^{+1}$.

The local algorithm $\mathcal{PA}$-MP$|_\triangleleft$ on processor v is composed of two *guarded actions*, which are atomic sets of instructions (actions) executed when a pre-condition (guard) is fulfilled (see Figure 2).

The guard of Rule $\mathcal{R}_1$ is true when a message m from u is received, while the guard of Rule $\mathcal{R}_2$ makes use of a timeout mechanism. So, our algorihtm is both message-driven (an action is executed when a new message is received) and timeout-driven (an action is executed when a timeout expires). In message passing systems, timeouts is required for stabilization purposes since [14] proved that no self-stabilizing algorithm could exist in message passing systems if no kind of timeout mechanism is available. The reason for this impossi-

```
$\mathcal{R}_1$   Upon receipt of a message $m$ sent by $u$:
          if $m \neq IN_v^u$, then
               Store$_v$($m$, $u$)
               Evaluate$_v$($\lhd_v$)
               Forward$_v$
          end if
$\mathcal{R}_2$   Upon timeout expiration:
          Evaluate$_v$($\lhd_v$)
          Forward$_v$
          reset the timeout
```

Figure 2: Local algorithm $\mathcal{PA}$-MP$|_{\lhd_v}$ on processor v.

bility result is that the system may start from an arbitrary global state where no messages are in transit, so if no node has a sending action that is triggered by a spontaneous timeout action, then the system is deadlocked.

Rule $\mathcal{R}_2$ is also used in case of message loss. In a typical implementation of our algorithm in an actual system, the timeout mechanism should be tuned accordingly to the loss rate of the communication links, in order that not too many spontaneous messages are emitted, and that the stabilization time remains reasonable. Tuning this timeout is clearly beyond the scope of this paper.

3.3 r-operators

An *infimum* (hereby called an *s-operator*) $\oplus$ over a set $\mathbb{S}$ is an associative, commutative and idempotent binary operator. Such an operator defines a partial order relation $\preceq_\oplus$ over the set $\mathbb{S}$ by $x \preceq_\oplus y$ if and only if $x \oplus y = x$ and then a strict order relation $\prec_\oplus$ by $x \prec_\oplus y$ if and only if $x \preceq_\oplus y$ and $x \neq y$.

It is generally assumed that there exists a greatest element on $\mathbb{S}$, denoted by $e_\oplus$, and verifying $x \preceq_\oplus e_\oplus$ for every $x \in \mathbb{S}$. Hence, the $(\mathbb{S}, \oplus)$ structure is an *Abelian idempotent semi-group* with $e_\oplus$ as identity element. The prefix *semi* means that the structure cannot be completed to obtain a group, because the law $\oplus$ is idempotent (see [3]).

When parameterized by such an s-operator $\oplus$, the $\mathcal{PA}$-MP parametric local algorithm converges. However, some counter examples show that it is not self-stabilizing [10]. Consider a loop with a single node initialized with 1 and using the operator min. The output of the node should always be 1. Now suppose that a fault introduces a 0 in the output register of the node (which is sent to itself). Then the node will never produce the correct result.

In [9], a distorted algebra — the r-algebra — is proposed. This algebra generalizes the Abelian idempotent semi-group, and still allows convergence of wave-like algorithms: the three basic properties (associativity, commutativity, idempotency) defining the s-operators are generalized using a mapping (usually denoted r). For instance, the binary operator $\diamond$ defined on the integers by $x \diamond y = x + 2y$ is not associative. However we have $x \diamond (y \diamond z) = (x \diamond y) \diamond 2z = x \diamond y \diamond 2z = x + 2y + 4z$ and $\diamond$ is *r-associative* with the mapping $x \mapsto 2x$.

The following definition summarizes the conditions of existence of the r-operators. The first one (right identity element) is classical. Here, the structure is not necessarily commutative, and only a right identity element is required. The second one (weak left cancellation) is very useful for allowing some simplifications in structures that do not admit inverses (such as idempotent semi-groups). It can be interpreted as follows: if there exists no element x in the definition set that does not agree with the fact that $y = z$, then $y = z$. Almost all useful operators are weak left cancellative, including the laws of groups (eg. addition on the integers) and of semi-groups (eg. minimum on the integers).

Definition 1 *The binary operator $\triangleleft$ on $\mathbb{S}$ is an r-*operator *if there exists a surjective mapping r called r-mapping, such that the following conditions are fulfilled:*
(i) *right identity element: $\exists e_\triangleleft \in \mathbb{S}, x \triangleleft e_\triangleleft = x$.*
(ii) *weak left cancellation: $\forall y, z \in \mathbb{S}$, $(\forall x \in \mathbb{S}, x \triangleleft y = x \triangleleft z) \Leftrightarrow (y = z)$*
(iii) *r-associativity: $\forall x, y, z \in \mathbb{S}, x \triangleleft (y \triangleleft z) = (x \triangleleft y) \triangleleft r(z)$;*
(iv) *r-commutativity: $\forall x, y \in \mathbb{S}, r(x) \triangleleft y = r(y) \triangleleft x$;*
*and **(v)** r-idempotency: $\forall x \in \mathbb{S}, r(x) \triangleleft x = r(x)$*

For example, the operator $\mathrm{minc}(x, y) = \min(x, y + 1)$ (for minimum and increment) is an r-operator on $\mathbb{Z} \cup \{+\infty\}$, with $+\infty$ its right identity element.

Given an r-operator $\triangleleft$, one can show that the r-mapping r is unique, and is an isomorphism of $(\mathbb{S}, \triangleleft)$. Moreover, the r-operator induces an s-operator on $\mathbb{S}$ by $x \triangleleft y = x \oplus r(y)$ (for instance, the r-operator minc induces the s-operator min). We also have $e_\oplus = e_\triangleleft$ and $r(e_\oplus) = e_\oplus$.

If no fault appears in the distributed system $\mathcal{S}$, our $\mathcal{PA}$-MP algorithm stabilizes when it is parameterized by any idempotent r-operator $\triangleleft$ [9]. Idempotent r-operators verify $x \preceq_\oplus r(x)$ for any $x \in \mathbb{S}$. This last property leads to the definition of *strict idempotency*, verified for instance by the r-operator minc:

Definition 2 *An r-operator $\triangleleft$ is* strictly idempotent *if, for any $x \in \mathbb{S} \setminus \{e_\oplus\}$, we have $x \prec_\oplus r(x)$.*

Note that, among others interesting properties, while it is not necessarily commutative, an r-operator $\triangleleft$ satisfies $\forall x, y, z \in \mathbb{S}$, $x \triangleleft y \triangleleft z = x \triangleleft z \triangleleft y$, which means that the result of the $\mathcal{PA}$-MP algorithm does not rely on any ordering of the neighborhood.

Finally, binary r-operators can be extended to accept any number of arguments. This is useful for our algorithm because a processor computes a result with one value per direct predecessor plus its own initialization value. An *n-ary r-operator* $\triangleleft$ consists in $n - 1$ binary r-operators based on the same s-operator, and we have, for any $x_0, \ldots, x_{n-1}$ in $\mathbb{S}$, $\triangleleft(x_0, \ldots, x_{n-1}) = x_0 \oplus r_1(x_1) \oplus \cdots \oplus r_{n-1}(x_{n-1})$. If all of these binary r-operators are (strictly) idempotent, the resulting n-ary r-operator is said (strictly) idempotent.

3.4 Hypotheses

In this section, we formalize some hypotheses, introduce some notations, and give basic lemmas that are used throughout the proofs.

Hypothesis 1 *In the distributed system $\mathcal{S}$, links may (fairly) lose, (finitely) duplicate, and (arbitrarily) reorder messages that are sent by neighboring processors. However, any message sent by u on the link (u, v) that is not lost is eventually received by v* (i.e. *no message may remain in a communication link forever).*

This is a weak hypothesis on link's reliability. However, the following lemma is immediate.

Lemma 1 *Let consider a communication link $(u, v) \in \mathcal{L}$. If the origin node u keeps sending the same message infinitely often, then this message is eventually received by the destination node v.*

Hypothesis 2 *In the distributed system $\mathcal{S}$ running the $\mathcal{PA}$-MP algorithm, any processor v runs the local algorithm defined in Figure 2 and parameterized by a strictly idempotent $(\delta v+1)$-ary r-operator. Moreover, all these r-operators are defined on the same set $\mathbb{S}$, and are based on the same s-operator $\oplus$, with $e_\oplus$ their common identity element.*

In other words, this hypothesis ensures a form of homogeneity in the distributed system we consider. The following lemma is a direct application of Hypothesis 2, Definition 1, and `Evaluate` function:

Lemma 2 *Let $\triangleleft_v$ be the r-operator used by processor v. Then the computation of the* `Evaluate`$_v(\triangleleft_v)$ *function can be rewritten as:*
$\triangleleft_v(\mathtt{IN}_v, \mathtt{IN}_v^{u_1}, \ldots, \mathtt{IN}_v^{u_k}) = \mathtt{IN}_v \oplus r_v^{u_1}(\mathtt{IN}_v^{u_1}) \oplus \cdots \oplus r_v^{u_k}(\mathtt{IN}_v^{u_k})$.

Hence, there is one r-mapping per communication link. We now define the composition of these mappings along a path ($\mathcal{X}_{u,v}$ denotes the set of all elementary paths from u to v).

Definition 3 *Let $P_{u_0,u_k} \in \mathcal{X}_{u_0,u_k}$ be a path from processor u_0 to processor u_k, composed of the edges (u_i, u_{i+1}) $(0 \leq i < k)$. Let r_{i+1}^i, $0 \leq i < k$, be the r-mapping associated to the link (u_i, u_{i+1}). The* r-path-mapping *of P_{u_0,u_k}, denoted by $r_{P_{u_0,u_k}}$, is defined by the composition of the r-mappings r_{i+1}^i, for $0 \leq i < k$: $r_{P_{u_0,u_k}} = r_k^{k-1} \circ \cdots \circ r_1^0$.*

Our proofs of correctness (Lemmas 7 and 12) assume that any result produced on a node with the `Evaluate`$_v(\triangleleft_v)$ function (see Lemma 2) is either the initial value of the node ($\mathtt{IN}_v$) or one of its incoming value transformed by an r-mapping ($r_v^{u_i}(\mathtt{IN}_v^{u_1})$). For this purpose, we admit that the order $\preceq_\oplus$ defines a total order. Note that with stronger nodes synchronization, such hypothesis is not necessary (see [11], where a proof for composite atomicity in a shared memory model is given).

Hypothesis 3 *The order relation $\preceq_\oplus$ is a total order relation: $\forall x, y \in \mathbb{S}$, either $x \preceq_\oplus y$ or $y \preceq_\oplus x$.*

Since the order $\preceq_\oplus$ is total, when it is clear from the context, in the remaining of the paper, we use "x is smaller than y" (or "y is larger than x") to denote $x \preceq_\oplus y$.

Hypothesis 4 *The set $\mathbb{S}$ is either finite, or any strictly increasing infinite sequence of values of $\mathbb{S}$ is unbounded (except by $e_\oplus$).*

Assuming Hypothesis 3, Hypothesis 4 specifies that the values used in the distributed system $\mathcal{S}$ can be, for instance, integers but not reals. Note that truncated reals (as in any computer implementation) are also convenient. Hypotheses 2 and 4 give the following lemma:

Lemma 3 *The set $\mathbb{S}$ is either finite or any r-mapping r used in $\mathcal{S}$ verifies: $\forall x \in \mathbb{S}\backslash\{e_\oplus\}, r(x) \prec_\oplus e_\oplus$.*

Hypothesis 5 *Each processor v admits at least one predecessor $u \in \Gamma_v^-$ such that $\mathtt{IN}_u \neq e_\oplus$, u is called a* non-null *processor.*

In the following, we denote by $\widehat{\mathtt{OUT}}_v$ the legitimate output of processor v. Moreover, for any processor v, any predecessor u of v and any configuration c, we denote by $\mathtt{OUT}_v(c)$ and $\mathtt{IN}_v^u(c)$ the value of the memories $\mathtt{OUT}_v$ and $\mathtt{IN}_v^u$ in the configuration c.

3.5 Our result

Our protocol is dedicated to static tasks. Such tasks (*e.g.*, the distance computation from a processor u) are defined by one output per processor v (*e.g.*, the distance from u to v), which is the legitimate output of v. With our $\mathcal{PA}$-MP algorithm, this means that, after finite time, each processor $v \in \mathcal{P}$ should contain this output (*e.g.*, $d(u,v)$) in its outgoing memory $\mathtt{OUT}_v$. To solve static tasks with the $\mathcal{PA}$-MP distributed algorithm, one must use an operator as parameter (*e.g.*, minc for distance computation) such that the distributed system $\mathcal{S}$ reaches the legitimate configurations and do not leave them thereafter (*i.e.*, any processor reaches and then conserves its legitimate output). In this paper, we prove that if the operator is used to parameterize the $\mathcal{PA}$-MP distributed algorithm, then it is self-stabilizing, according to the hypotheses of § 3.4.

Let us define the legitimate outputs of the processor using the r-operators that parameterize the $\mathcal{PA}$-MP algorithm. For instance, to solve the distance computation problem, we state $\mathbb{S} = \mathbb{N} \cup \{+\infty\}$, and each local algorithm is parameterized by the minc r-operator (see § 3.3). All processors v verify $\mathtt{IN}_v = +\infty$ except a non null processor u verifying $\mathtt{IN}_u = 0$ (0 is absorbing while $+\infty$ is the identity element for minc). Each r-path-mapping adds its length to its argument (*i.e.*, $r_P(x) = x + \text{length}(P)$), and we have:

$$d(u,v) = \min\left(\mathtt{IN}_v, \min_{w\in\Gamma_v^-, P_{w,v}\in\mathcal{X}_{w,v}} \left\{r_{P_{w,v}}\left(\mathtt{IN}_w\right)\right\}\right)$$

We now define the legitimate output of a processor v in the general case.

Definition 4 (Legitimate output) *The* legitimate output *of processor v is:*

$$\widehat{\mathtt{OUT}}_v = \mathtt{IN}_v \oplus \bigoplus_{u\in\Gamma_v^-, P_{u,v}\in\mathcal{X}_{u,v}} r_{P_{u,v}}\left(\mathtt{IN}_u\right)$$

The following lemma is given by Lemma 3, Hypothesis 5 and Definition 4; it is used for proving Theorem 1.

Lemma 4 *The set $\mathbb{S}$ is either finite or any processor $v \in \mathcal{P}$ verifies: $\widehat{\texttt{OUT}}_v \prec_\oplus e_\oplus$.*

Now we defined $\widehat{\texttt{OUT}}_v$, we define the set of legitimate configurations $L \subset \mathcal{C}$ of the protocol $\mathcal{PA}$-MP (see Section 3 and Figure 2):

Definition 5 (Legitimate configuration) *For any configuration $c \in L$, for any processor $v \in \mathcal{P}$, $\texttt{OUT}_v(c) = \widehat{\texttt{OUT}}_v$.*

Finally, after defining the distributed system $\mathcal{S}$, the generic algorithm $\mathcal{PA}$-MP, the r-operators used as parameters and some Hypotheses, we can express the main result of this paper as follows, which is proved in the following section:

Theorem 1 *Algorithm $\mathcal{PA}$-MP parameterized by any strictly idempotent r-operator is self-stabilizing in directed message passing networks, despite fair loss, finite duplication and reordering of messages.*

The message passing model that we consider leads to hard difficulties (compared for instance to shared memory model [10]). Indeed, with this model it is possible that an initially wrong message remains in a link for quite a long (finite) time (*e.g.* after several new messages have been exchanged) and then is delivered to cause havoc in the system. Note that to reuse [10] in unreliable message passing systems, a self-stabilizing data link protocol is required, yet no such data link protocol exists in unidirectional networks. So, our approach is the first to date to support multiple metrics in (realistic) unreliable unidirectional networks. We hereby give the main proof arguments. Details are provided in Section 4.

Scketch of proof: Despite weak hypotheses on the communication capabilities of every link (u, v), and possible transient failures that could corrupt data in links or nodes communication buffers $\texttt{OUT}_u$ and $\texttt{IN}_v^u$, we have to prove that eventually any input value read by v in $\texttt{IN}_v^u$ has effectively been sent by u. Even though this is true, it does not imply that a value sent by u will be received by v. Hence, a legitimate value sent by u could be lost in (u, v), while the inputs of u that were used to produce it disappeared, either because of transient failures, or simply because they were overwritten by other incoming values. This means that legitimate values could completely be removed from $\mathcal{S}$.

We actually have to prove that a value received by v on (u, v) has been sent by u *after* a given configuration. This configuration is chosen such that the value of u fulfills some predicates. One of those predicates is that this value has been built using incoming values of u sent by its predecessors *after* a given configuration. This permits to use recursivity along paths of the network.

By weak fairness, any processor v calls `Evaluate` for updating its output $\texttt{OUT}_v$ using its inputs. By properties of the r-operators, and using the total order Hypothesis (Hyp. 3), this output is either built with $\texttt{IN}_v$ or with a received value, say $\texttt{IN}_v^u$. After the last transient failure, and since duplications are finite on the link (u, v), any value received by v has been sent by u. Since every perturbation on the link is finite, there is a finite number of configurations between the sending of the value by u and its receipt by v. Thus, if we consider a configuration that is far enough in the execution, v must have updated its output using a value received by u after u has itself updated its output too. This way, we can prove that any output is smaller or equal than the legitimate value, which means that every large unlegitimate value eventually disappears from the network.

To complete the proof of correctness, we still have to prove that every processor v may not remain with a smaller value than its legitimate one. Suppose this is the case, then by reusing a recursive reasoning, we obtain an infinite path of processors, such that their outputs are strictly increasing along the path (by the strict idempotency property of the r-operators). Since such a path does not exists in the network (that is finite), it is a cycle. This means that, successive outputs of v increase without ever reaching its legitimate value. That contradicts Hypothesis 4. □

3.6 Example

Some r-operators have been proposed to compute the minimum distance tree and forest, the shortest path tree and forest, the best reliable paths from some transmitters, the depth first search tree... More complex applications [12] have also been proposed by combining several r-operators.

For instance, when the local algorithms are parameterized by the minc r-operator, the system stabilizes to a minimum distance tree when all the node are initialized with $e_{\mathrm{minc}} = +\infty$ except one (the root) initialized with 0 (see Figure 3).

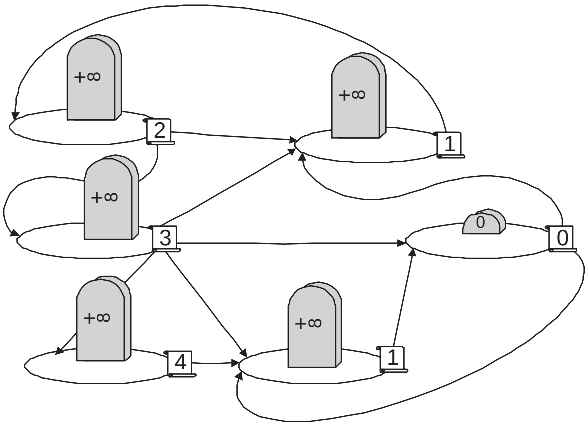

Figure 3: The minc r-operator on each node leads to a minimum distance tree computation in a unreliable unidirectional network.

4 Proof of Correctness

This section is divided into six parts. First, we give basic results related to the operators. Second, we prove that eventually the output of each processors is updated using its inputs. Third, we show that eventually each received message was sent in the past. Fourth, we prove that each processor's output is upper bounded. Fifth, we prove that each processor eventually reaches its legitimate value. Finally, we present complexity results regarding our distributed protocol.

4.1 Properties of the operators

Any r-operator $\triangleleft$ defined on $\mathbb{S}$ induces an s-operator $\oplus$ on $\mathbb{S}$ by $x \oplus r(y) = x \triangleleft y$. Since the s-operator defines an order relation $\preceq_{\oplus}$ by $x \preceq_{\oplus} y \equiv x \oplus y = x$, the Lemma 5 holds. Since r is an homomorphism of $(\mathbb{S}, \oplus)$, the Lemma 6 holds.

Lemma 5 *For all $x, y, z \in \mathbb{S}$, if $x \oplus y = z$ then $z \preceq_{\oplus} x$ and $z \preceq_{\oplus} y$.*

Lemma 6 *For all $x, y \in \mathbb{S}$, if $x \preceq_{\oplus} y$, then $r(x) \preceq_{\oplus} r(y)$.*

4.2 Outputs eventually result from computations

We begin by defining some predicates on configurations.

Definition 6 *Let P_{0a}, P_{0b} and P_{0c} be predicates on configurations $c \in \mathcal{C}$:*

$$\begin{array}{lcl} P_{0a}(c) & \equiv & \forall v \in \mathcal{P}, \quad \mathtt{OUT}_v(c) \preceq_{\oplus} \mathtt{IN}_v \\ P_{0b}(c) & \equiv & \forall v \in \mathcal{P}, \forall u \in \Gamma_v^{-1}, \quad \mathtt{OUT}_v(c) \preceq_{\oplus} r_v^u(\mathtt{IN}_v^u(c)) \\ P_{0c}(c) & \equiv & \forall v \in \mathcal{P}, \forall u \in \Gamma_v^{-1}, \quad \mathtt{OUT}_v(c) = \mathtt{IN}_v \\ & & \vee\, \mathtt{OUT}_v(c) = r_v^u(\mathtt{IN}_v^u(c)) \end{array}$$

Now, the set $Q_0 \subset \mathcal{E}$ includes executions where processors eventually update their output. Every execution e of Q_0 reaches a configuration c_{i_0} such that any subsequent configuration c_j satisfies Predicates P_{0a}, P_{0b} and P_{0c}.

Definition 7 *Let $Q_0 \subset \mathcal{E}$ be the set of executions such that:*

$$\forall e \in Q_0, \quad \exists c_{i_0} \in e_{|\mathcal{C}}, \forall c_j \in e_{|\mathcal{C}} \text{ with } i_0 \leq j, \\ P_{0a}(c_j) \wedge P_{0b}(c_j) \wedge P_{0c}(c_j)$$

We now prove that, thanks to weak fairness hypothesis, any execution of $\mathcal{E}$ is in Q_0.

Lemma 7 *Every execution of the $\mathcal{PA}$-MP algorithm in the distributed system $\mathcal{S}$ is in Q_0.*

Proof: Let $e \in \mathcal{E}$ be an execution. By weak fairness, every processor $v \in \mathcal{P}$ eventually executes a rule. By definition of $\mathcal{PA}$-MP (see Figure 2), any execution of either rule at some node v processes $\mathtt{Evaluate}_v(\triangleleft_v)$. Then, for any processor $v \in \mathcal{P}$, there exists a configuration $c_{i_v} \in e_{|\mathcal{C}}$ where processor v satisfies $\mathtt{OUT}_v(c_{i_v}) = \triangleleft_v (\mathtt{IN}_v, \mathtt{IN}_v^{u_1}(c_{i_v}), \ldots, \mathtt{IN}_v^{u_{\delta v}}(c_{i_v}))$.

By Lemma 2, we have $\mathtt{OUT}_v(c_{i_v}) = \mathtt{IN}_v \oplus r_v^{u_1}(\mathtt{IN}_v^{u_1}(c_{i_v})) \oplus \cdots \oplus r_v^{u_{\delta v}}(\mathtt{IN}_v^{u_{\delta v}}(c_{i_v}))$. Then, by Lemma 5, we have $\mathtt{OUT}_v(c_{i_v}) \preceq_{\oplus} \mathtt{IN}_v$ and $\mathtt{OUT}_v(c_{i_v}) \preceq_{\oplus} r_v^u(\mathtt{IN}_v^u)$ for any direct-predecessor u of v. Hence, both $P_{0a}(c_{i_v})$ and $P_{0b}(c_{i_v})$ hold. Now, since $\preceq_{\oplus}$ defines a total order relation (Hypothesis 3), either $\mathtt{OUT}_v(c_{i_v}) = \mathtt{IN}_v$ or $\mathtt{OUT}_v(c_{i_v}) = r_v^u(\mathtt{IN}_v^u(c_{i_v}))$ for at least one predecessor u of v. This gives $P_{0c}(c_{i_0})$ with $i_0 = \max_{v \in \mathcal{P}} i_v$.

Since any action of v executed upon receipt of a message or upon timeout expiration calls $\mathtt{Evaluate}$, any subsequent configuration satisfies Predicates P_{0a} to P_{0c}. □

4.3 Eventually, received messages were previously sent

We define the set Q_1 as the subset of executions $\mathcal{E}$ for which any received value has actually been sent in the past. All executions e of Q_1 reach a configuration c_{i_1} such that, for any subsequent configuration c_j and any communication link (u, v), there exists a configuration $c_{j_{uv}}$ in which v sent the value contained in $\mathtt{IN}_v^u$ in configuration c_j.

Definition 8 *Let $Q_1 \subset \mathcal{E}$ be the set of executions that satisfy:*

$$\forall e \in Q_1, \qquad \exists c_{i_1} \in e_{|\mathcal{C}} \quad \begin{cases} \forall c_j \in e_{|\mathcal{C}} \text{ with } i_1 \leq j, \forall (u,v) \in \mathcal{L}, \\ \exists c_{j_{uv}} \in e_{|\mathcal{C}} \text{ with } j_{uv} \leq j, \quad \mathtt{OUT}_u(c_j) = \mathtt{IN}_v^u(c_{j_{uv}}) \end{cases}$$

We now prove that, thanks to Hypothesis 1 related to the properties of the communications links, any execution is in Q_1.

Lemma 8 *Every execution of the $\mathcal{PA}$-MP algorithm in the distributed system $\mathcal{S}$ is in Q_1.*

Proof: Let $e \in \mathcal{E}$ be an execution, and consider two processors u and v such that (u, v) is a communication link of $\mathcal{L}$. By definition of $\mathcal{PA}$-MP, processor v sends the value of its $\mathtt{OUT}_v$ variable infinitely often to each of its direct successors. By Hypothesis 1, every message that is not lost is eventually delivered. Moreover, every message may be duplicated only a finite number of times. It follows that, after a finite amount of time, only messages that were sent by v are received by all of its direct successors. Hence, there exists a configuration $c_j \in e_{|\mathcal{C}}$ where the incoming value in $\mathtt{IN}_v^u$ has actually been sent by u in a previous configuration $c_{j_{uv}}$:

$$\mathtt{IN}_v^u(c_j) = \mathtt{OUT}_u(c_{j_{uv}}) \text{ with } j_{uv} \leq j \tag{1}$$

After all initial erroneous messages between u and v have been received (including duplicates), and after a configuration where the above property holds, this property remains thereafter on this link. Since all links conform to the same hypotheses, there exists a configuration $c_{i_1} \in e_{|\mathcal{C}}$ where the property holds (and remains so thereafter) for any communication link. We conclude that $e \in Q_1$. □

Note that this lemma does not indicate that any sent value is eventually received. Indeed, it may happen that a message is lost while traversing a link, and the variable it was built with is erased by a new value. Then, any re-sending would not provide the original value, that would not be received again. We now generalize the notation we introduced in the previous proof.

Definition 9 *Let us consider an incoming value $\mathtt{IN}_v^u(c_j)$ on processor v in the configuration c_j. Then we denote by $c_{j_{uv}}$ the configuration in which the value $\mathtt{IN}_v^u(c_j)$ has been sent by u, provided that this configuration exists.*

The previous lemma indicates that, for any execution $e \in \mathcal{E}$, there exists a configuration c_{i_1} from which $c_{j_{uv}}$ exists for any subsequent configuration c_j ($i_1 \leq j$), and any communication link (u, v). However, as captured in Figure 4, the definition of Q_1 gives no guarantees about $c_{j_{uv}}$ appearing after configuration c_{i_1} (that is $i_1 \leq j_{uv}$).

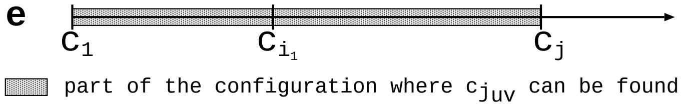

Figure 4: According to Q_1, configuration $c_{j_{uv}}$ exists but could appear before c_{i_1}.

We now introduce additional sets of executions. The following definition, illustrated in Figure 5, indicates that, for any execution in Q_{1b}, from a given configuration $c_{i_{1b}}$, any given configuration c_i admits a configuration $c_{i'}$ such that any configuration $c_{j_{uv}}$ (with $i' \leq j$) appeared after c_i (*i.e.*, $i \leq j_{uv}$).

Definition 10 *Let $Q_{1b} \subset \mathcal{E}$ be the set of executions that satisfy:*

$$\forall e \in Q_{1b}, \qquad \exists c_{i_{1b}} \in e_{|\mathcal{C}}$$
$$\begin{cases} \forall c_i \in e_{|\mathcal{C}} \textit{ with } i_{1b} \leq i, \exists c_{i'} \in e_{|\mathcal{C}} \textit{ with } i \leq i', \\ \forall c_j \in e_{|\mathcal{C}} \textit{ with } i' \leq j, \forall (u,v) \in \mathcal{L}, \\ c_{j_{uv}} \in e_{|\mathcal{C}} \wedge \quad i \leq j_{uv} \leq j \end{cases}$$

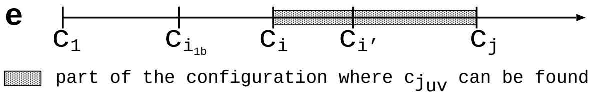

Figure 5: According to Q_{1b}, from a configuration $c_{i_{1b}}$, configurations $c_{j_{uv}}$ can be found later than any given configuration c_i.

We show now that, thanks to weak fairness, every execution is in Q_{1b}.

Lemma 9 *Every execution of the $\mathcal{PA}$-MP algorithm in the distributed system $\mathcal{S}$ is in Q_{1b}.*

Proof: Let $e \in \mathcal{E}$ be an execution that is not in Q_{1b}. From Lemma 8, e is in Q_1 and, from a configuration $c_{i_1} \in e_{|\mathcal{C}}$, for every configuration c_j and every link (u,v), the configuration $c_{j_{uv}}$ exists. Now, let us consider configurations c_i, $c_{i'}$ and c_j in $e_{|\mathcal{C}}$ such that $i_1 \leq i \leq i' \leq j$. If $e \notin Q_{1b}$, then configuration $c_{j_{uv}}$ always appears before c_i, even if $c_{i'}$ (and then c_j) is as far as possible from c_i (see Figure 6). This means that the values produced by processor u after $c_{j_{uv}}$ were never received, that contradicts Lemma 1.

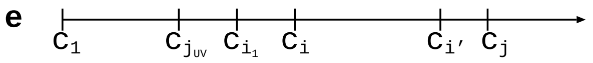

Figure 6: If $e \notin Q_{1b}$, the configuration $c_{j_{uv}}$ always appears before c_i.

□

4.4 Outputs are eventually smaller than (or equal to) legitimate values

Let us begin by defining two predicates P_2 and P_{2b} on configurations. If $P_2(c)$ holds, then, in configuration c, each processor is smaller than all initial values of its predecessors increased by some r-mappings (more precisely, for any processor v and any of its direct-predecessors u, the output of v is smaller than the initial value of u transformed by the r-path-mapping $r_{P_{u,v}}$ of the path $P_{u,v}$ from u to v). If $P_{2b}(c)$ holds, then, in the configuration c, the output of each processor v is smaller (in the sense of $\oplus$) than its legitimate output.

Definition 11 *Let P_2 and P_{2b} be predicates on configurations $c \in \mathcal{C}$:*

$$\begin{aligned} P_2(c) \quad &\equiv \quad \forall v \in \mathcal{P}, \forall u \in \Gamma_v^-, \forall P_{u,v} \in \mathcal{X}_{u,v}, \\ & \mathtt{OUT}_v(c) \preceq_\oplus r_{P_{u,v}}(\mathtt{IN}_u) \\ P_{2b}(c) \quad &\equiv \quad \forall v \in \mathcal{P}, \quad \mathtt{OUT}_v(c) \preceq_\oplus \widehat{\mathtt{OUT}}_v \end{aligned}$$

We now define two sets of executions Q_2 and Q_{2b}. If an execution e is in Q_2 (resp· Q_{2b}), then there exists a configuration in e from which every configuration satisfies P_2 (resp. P_{2b}).

Definition 12 *Let Q_2 and Q_{2b} be two subsets of $\mathcal{E}$:*

$$\begin{aligned} &\forall e \in Q_2, \quad \exists c_{i_2} \in e_{|\mathcal{C}}, \quad \forall c_j \in e_{|\mathcal{C}} \textit{ with } i_2 \leq j, \quad P_2(c_j) \\ &\forall e \in Q_{2b}, \quad \exists c_{i_{2b}} \in e_{|\mathcal{C}}, \quad \forall c_j \in e_{|\mathcal{C}} \textit{ with } i_{2b} \leq j, \quad P_{2b}(c_j) \end{aligned}$$

We now prove that, first every execution of $\mathcal{E}$ is in Q_2, and then that every execution of $\mathcal{E}$ is in Q_{2b}. This means that, while the processor's outputs can be larger than the legitimate values in the beginning of an execution, each processor eventually produces some outputs that are smaller than or equal to its legitimate value. In other terms, any erroneous values that are larger than legitimate values eventually disappear from $\mathcal{S}$.

Lemma 10 *Every execution of the $\mathcal{PA}$-MP algorithm in the distributed system $\mathcal{S}$ is in Q_2.*

Proof: Let $e \in \mathcal{E}$ be an execution, and let us consider a processor $v_0 \in \mathcal{P}$, and one of its direct-predecessor $v_1 \in \Gamma_{v_0}^{-1}$. By Lemma 7, e is in Q_0. Then, there exists a configuration $c_{i_0} \in e_{|\mathcal{C}}$ such that, for any subsequent configuration $c_{j_{v_0}} \in e_{|\mathcal{C}}$ ($i_0 \leq j_{v_0}$), Predicate $P_{0b}(c_{j_{v_0}})$ is satisfied: $\mathtt{OUT}_{v_0}(c_{j_{v_0}}) \preceq_\oplus r_{v_0}^{v_1}(\mathtt{IN}_{v_0}^{v_1}(c_{j_{v_0}}))$.

Since $e \in Q_1$, the above configuration $c_{j_{v_0}}$ can be chosen after c_{i_1} in e (*i.e.*, $i_0 \leq j_{v_0}$ and $i_1 \leq j_{v_0}$) so that there exists a configuration $c_{j_{v_1v_0}} \in e_{|\mathcal{C}}$ that appears before $c_{j_{v_0}}$ (*i.e.*, $j_{v_1v_0} \leq j_{v_0}$) satisfying: $\mathtt{OUT}_{v_1}(c_{j_{v_1v_0}}) = \mathtt{IN}_{v_0}^{v_1}(c_{j_{v_0}})$. This gives:

$$\mathtt{OUT}_{v_0}(c_{j_{v_0}}) \preceq_\oplus r_{v_0}^{v_1}\left(\mathtt{OUT}_{v_1}(c_{j_{v_1v_0}})\right) \tag{2}$$

Since $e \in Q_{1b}$, it is possible to choose configuration $c_{j_{v_0}}$ in $e_{|\mathcal{C}}$ in order to ensure that $c_{j_{v_1v_0}}$ appears *after* c_{i_0}. Hence, without loss of generality, we can state $i_0 \leq j_{v_1v_0}$ and thus $P_{0a}(c_{j_{v_1v_0}})$ holds. This means that $\mathtt{OUT}_{v_1}(c_{j_{v_1v_0}}) \preceq_\oplus \mathtt{IN}_{v_1}$, and, from Lemma 6, we have: $r_{v_0}^{v_1}(\mathtt{OUT}_{v_1}(c_{j_{v_1v_0}})) \preceq_\oplus r_{v_0}^{v_1}(\mathtt{IN}_{v_1})$.

Finally, we obtain the following relation, that remains true for configurations that appear after $c_{j_{v_0}}$:

$$\mathtt{OUT}_{v_0}(c_{j_{v_0}}) \preceq_{\oplus} r_{v_0}^{v_1}(\mathtt{IN}_{v_1}) \tag{3}$$

and this result remains true hereafter.

To iterate the above reasoning from vertex v_1 (instead of v_0) at configuration $c_{j_{v_1v_0}}$ (instead of $c_{j_{v_0}}$), we must ensure that $c_{j_{v_1v_0}}$ appears after c_{i_0} (to use Q_0) and after c_{i_1} (to use Q_1). Yet using the fact that $e \in Q_{1b}$, the configuration $c_{j_{v_0}}$ can be chosen as far as necessary in e in order to ensure that the related configuration $c_{j_{v_1v_0}}$ happens *after* the configurations c_{i_0} and c_{i_1} (see Figure 5). Hence, for any path $v_k, \ldots, v_0$, there exist some configurations $c_{j_{v_kv_{k-1}}}, \ldots, c_{j_{v_1,v_0}}, c_{j_{v_0}}$ such that the following relations (obtained from Equations 2 and 3) remain true for the rest of the execution:

$$\begin{array}{rccc}
 & \mathtt{OUT}_{v_0}(c_{j_{v_0}}) & \preceq_{\oplus} & r_{v_0}^{v_1}(\mathtt{OUT}_{v_1}(c_{j_{v_1v_0}})) \\
\wedge & \mathtt{OUT}_{v_0}(c_{j_{v_0}}) & \preceq_{\oplus} & r_{v_0}^{v_1}(\mathtt{IN}_{v_1}) \\
 & \mathtt{OUT}_{v_1}(c_{j_{v_1v_0}}) & \preceq_{\oplus} & r_{v_1}^{v_2}(\mathtt{OUT}_{v_1}(c_{j_{v_2v_1}})) \\
\wedge & \mathtt{OUT}_{v_1}(c_{j_{v_1v_0}}) & \preceq_{\oplus} & r_{v_1}^{v_2}(\mathtt{IN}_{v_2}) \\
 & \vdots & \preceq_{\oplus} & \vdots \\
 & \mathtt{OUT}_{v_k}(c_{j_{v_kv_{k-1}}}) & \preceq_{\oplus} & r_{v_k}^{v_{k-1}}(\mathtt{OUT}_{v_0}(c_{j_{v_1v_0}})) \\
\wedge & \mathtt{OUT}_{v_k}(c_{j_{v_kv_{k-1}}}) & \preceq_{\oplus} & r_{v_k}^{v_{k-1}}(\mathtt{IN}_{v_{k-1}})
\end{array} \tag{4}$$

Then, for any predecessor v_k of v_0 and any path $P_{v_kv_0} \in \mathcal{X}_{v_k,v_0}$ from v_k to v_0, there exists a configuration $c_{j_{v_0}}$ such that the following remains true in any subsequent configuration: $\mathtt{OUT}_{v_0}(c_{j_{v_0}}) \preceq r_{P_{v_kv_0}}(\mathtt{IN}_{v_k})$. Hence there exists a configuration c_{i_2} reached after all configurations $c_{j_{v_0}}$ (for any processor $v_0 \in \mathcal{P}$) and such that, for any further configuration c_j (*i.e.*, $i_{2b} \leq j$), we have $P_2(c_j)$. This gives the lemma. □

Lemma 11 *Every execution of the $\mathcal{PA}$-MP algorithm in the distributed system $\mathcal{S}$ is in Q_{2b}.*

Proof: Let us consider an execution $e \in \mathcal{E}$. Since $e \in Q_2$, there exists a configuration $c_{i_2} \in e_{|\mathcal{C}}$ such that, for any subsequent configuration $c_j \in e_{|\mathcal{C}}$ (*i.e.*, $i_2 \leq j$), $P_2(c_j)$ holds:

$$\forall v \in \mathcal{P}, \forall u \in \Gamma_v^-, \forall P_{u,v} \in \mathcal{X}_{u,v}, \quad \mathtt{OUT}_v(c_j) \preceq_{\oplus} r_{P_{u,v}}(\mathtt{IN}_u)$$

Then, we have:

$$\forall v \in \mathcal{P}, \quad \mathtt{OUT}_v(c_j) \preceq_{\oplus} \bigoplus_{u \in \Gamma_v^-, P_{u,v} \in \mathcal{X}_{u,v}} r_{P_{u,v}}(\mathtt{IN}_u)$$

Since $e \in Q_1$, some of these configurations c_j also satisfy predicate P_{0a}. Without loss of generality, we assume that $P_{0a}(c_j)$ holds: $\mathtt{OUT}_v(c_j) \preceq_{\oplus} \mathtt{IN}_v$. Hence, we have:

$$\forall v \in \mathcal{P}, \quad \mathtt{OUT}_v(c) \preceq_{\oplus} \mathtt{IN}_v \oplus \bigoplus_{u \in \Gamma_v^-, P_{u,v} \in \mathcal{X}_{u,v}} r_{P_{u,v}}(\mathtt{IN}_u)$$

This ends the proof, by Definition 4. □

4.5 Legitimate values are eventually reached

Let us begin by defining a predicate on system configurations.

Definition 13 *Let P_3 be a predicate on configurations $c \in \mathcal{C}$:*

$$P_3(c) \quad \equiv \quad \forall v \in \mathcal{P}, \quad \texttt{OUT}_v(c) = \widehat{\texttt{OUT}}_v$$

We now define the set of executions Q_3, that corresponds to executions of $\mathcal{E}$ for which every processor eventually reach its legitimate value: all executions of Q_3 reach a configuration c_{i_3} such that, for any subsequent configuration c_j, the outputs of every processor v in c_j are equal to their legitimate values.

Definition 14 *Let $Q_3 \subset \mathcal{E}$ be the set of executions that satisfy:*

$$\forall e \in Q_3, \quad \exists c_{i_3} \in e_{|\mathcal{C}}, \forall c_j \in e_{|\mathcal{C}}, \textit{ with } i_3 \leq j, \quad P_3(c)$$

We now prove that any execution is in Q_3.

Lemma 12 *Every execution of the $\mathcal{PA}$-MP algorithm in the distributed system $\mathcal{S}$ is in Q_3.*

Proof: Let $e \in \mathcal{E}$ be an execution, and suppose that $e \notin Q_3$. Since $\preceq_\oplus$ defines a total order (Hypothesis 3), we have:

$$\begin{array}{l} \forall c_{i_3} \in e_{|\mathcal{C}}, \quad \exists c_j \in e_{|\mathcal{C}}, \text{ with } i_3 \leq j, \\ \exists v \in \mathcal{P}, \quad \widehat{\texttt{OUT}}_v \prec_\oplus \texttt{OUT}_v(c_j) \quad \vee \quad \texttt{OUT}_v(c_j) \prec_\oplus \widehat{\texttt{OUT}}_v \end{array} \tag{5}$$

By Lemma 11, e is in Q_{2b} and there exist some configurations c_j that satisfy both $i_3 \leq j$ and $i_{2b} \leq j$, so that $\texttt{OUT}_v(c_j) \preceq_\oplus \widehat{\texttt{OUT}}_v$. Hence, Equation 5 becomes:

$$\begin{array}{l} \forall c_{i_3} \in e_{|\mathcal{C}}, \quad \exists c_j \in e_{|\mathcal{C}}, \text{ with } i_3 \leq j, \\ \exists v \in \mathcal{P}, \quad \texttt{OUT}_v(c_j) \prec_\oplus \widehat{\texttt{OUT}}_v \end{array} \tag{6}$$

By Definition 4 and Lemma 5, we have $\widehat{\texttt{OUT}}_v \preceq_\oplus \texttt{IN}_v$. This gives $\texttt{OUT}_v(c_j) \prec_\oplus \texttt{IN}_v$. Since $e \in Q_0$, there exist some configurations $c_j \in e_{|\mathcal{C}}$ satisfying both Equation 6 and $P_{0c}(c_j)$, that is $i_0 \leq j$. Without loss of generality, we suppose that $P_{0c}(c_j)$ holds: $\exists u \in \Gamma_v^{-1}, \texttt{OUT}_v(c_j) = r_v^u(\texttt{IN}_v^u(c_j))$.

As $\texttt{OUT}_v(c_j) \prec_\oplus \widehat{\texttt{OUT}}_v$, we have $r_v^u(\texttt{IN}_v^u(c_j)) \neq e_\oplus$. Since $r_v^u(e_\oplus) = e_\oplus$ (see § 3.3), we have $\texttt{IN}_v^u(c_j) \neq e_\oplus$. Then, by Definition 2, we have $\texttt{IN}_v^u(c_j) \prec_\oplus r_v^u(\texttt{IN}_v^u(c_j))$ and finally $\texttt{IN}_v^u(c_j) \prec_\oplus \texttt{OUT}_v(c_j)$. Hence, the following holds: $\exists u \in \Gamma_v^{-1}, \texttt{IN}_v^u(c_j) \prec_\oplus \texttt{OUT}_v(c_j)$.

By Lemma 8, $e \in Q_1$, and there exists some configuration c_j that satisfy $i_1 \leq j$ (as well as $i_4 \leq j$, $i_{2b} \leq j$ and $i_0 \leq j$) and for which configuration $c_{j_{uv}}$ exists in e and verifies $\texttt{OUT}_u(c_{j_{uv}}) = \texttt{IN}_v^u(c_j)$. Then $\texttt{OUT}_u(c_{j_{uv}}) \prec_\oplus \texttt{OUT}_v(c_j) \prec_\oplus \widehat{\texttt{OUT}}_v$. This means that at least one of the direct-predecessors u of v verifies $\texttt{OUT}_u(c_{j_{uv}}) \prec_\oplus \widehat{\texttt{OUT}}_u \vee \widehat{\texttt{OUT}}_u \prec_\oplus \texttt{OUT}_u(c_{j_{uv}})$ (indeed, if all predecessors of v reached and hold their legitimate value, then v would reach its legitimate value too). Hence, Equation 6 becomes:

$$\begin{array}{l}\forall c_{i_3} \in e_{|\mathcal{C}}, \quad \exists c_j \in e_{|\mathcal{C}}, \text{ with } i_3 \leq j, \\ \exists u, v \in \mathcal{P} \text{ with } u \in \Gamma_v^{-1}, \quad \exists c_{j_{uv}} \in e_{|\mathcal{C}}, \text{ with } j_{u,v} \leq j \\ \left(\texttt{OUT}_u(c_{j_{uv}}) \prec_\oplus \widehat{\texttt{OUT}}_u \quad \vee \quad \widehat{\texttt{OUT}}_u \prec_\oplus \texttt{OUT}_u(c_{j_{uv}})\right) \\ \wedge \quad \texttt{OUT}_u(c_{j_{uv}}) \prec_\oplus \texttt{OUT}_v(c_j) \prec_\oplus \widehat{\texttt{OUT}}_v \end{array} \tag{7}$$

To iterate the above argument from processor u instead of v, and from configuration $c_{j_{uv}}$ instead of c_j, we argue that $i_0 \leq j_{uv}$, $i_1 \leq j_{uv}$, and $i_{2b} \leq j_{uv}$. By Lemma 9, e is in Q_{1b}. This means that configurations c_{i_3} in the above equation can be chosen so that every configurations $c_{j_{uv}}$ appear *after* configurations c_{i_0}, c_{i_1} and $c_{i_{2b}}$ (see Figure 5). This allows to re-use the above reasoning with configuration $c_{j_{uv}}$ instead of c_j.

By iterating the above arguments, and since the network is finite, we exhibit a cycle of nodes and a set of configurations $c_{j_0}, c_{j_1} \ldots$ appearing after c_{i_4} in e such that, for a node w in the cycle, we have:

$$\texttt{OUT}_w(c_{j_0}) \prec_\oplus \texttt{OUT}_w(c_{j_1}) \prec_\oplus \ldots \prec_\oplus \widehat{\texttt{OUT}}_w \tag{8}$$

Using the fact that $e \in Q_{1b}$, this can be found after any configuration c_{i_4} in the execution e. This means that, regardless of configuration c_{i_4}, there exist subsequent configurations $c_{j_0}, \ldots c_{j_1}$, such that $\widehat{\texttt{OUT}}_w$ increases strictly without reaching its legitimate value. We then exhibit a strictly increasing sequence of values of $\mathbb{S}$ that never reach $\widehat{\texttt{OUT}}_w$. This is impossible if $\mathbb{S}$ is finite. If $\mathbb{S}$ is infinite, then Lemma 4 gives $\widehat{\texttt{OUT}}_w \prec_\oplus e_\oplus$. The sequence of values is then upper bounded, that contradicts Hypothesis 4. Hence, $e \in Q_3$. □

4.6 Complexity

In the convergence part of the proof, we only assumed that computations were maximal, and that message loss, duplication and desequencing could occur. In order to provide an upper bound on the stabilization time for our algorithm, we assume strong synchrony between processors and a reliable communication medium between nodes. Note that these assumptions are used for complexity results only, since our algorithm was proved correct even in the case of asynchronous unfair computations with link intermittent failures. In the following, D denotes the network diameter.

In order to give an upper bound on the space and time requirements, we assume that the set $\mathbb{S}$ is finite, and that $|\mathbb{S}|$ denotes its number of elements. This assumption is used for complexity results only, since our algorithm was proved to be correct even in the case when $\mathbb{S}$ is infinite. Note that in any implementation the set of possible values is finite, and if the memories $\texttt{IN}_v$ and $\texttt{OUT}_v$ of each node v contains n bits, then $|\mathbb{S}| = 2^n$.

The space complexity result is immediately given by the assumptions made when writing Algorithm $\mathcal{PA}$-MP.

Lemma 13 (Space Complexity) *Each processor $v \in \mathbb{S}$ holds $(\delta v + 1) \times \log_2(|\mathbb{S}|)$ bits.*

Proof: Each processor v has δv local variables that hold the value of the last message sent by the corresponding direct predecessor, and one register used to communicate with its direct descendants. Each of these local variables may hold a value in a finite set $\mathbb{S}$, then need $\log_2(|\mathbb{S}|)$ bits. Note that the constant stored in ROM is not taken into account in this result. □

Lemma 14 (Time Complexity) *Assuming a synchronous system $\mathcal{S}$, the stabilization time is $O(D+|\mathbb{S}|)$.*

Proof: We define ϕ as the function that returns the index of a given element of $\mathbb{S}$. This index always exists since $\mathbb{S}$ is ordered by a total order relation. The signature of ϕ is as follows:

$$\begin{array}{llll} \phi: & \mathbb{S} & \rightarrow & \mathbb{N} \\ & s & \mapsto & \phi(s) \end{array}$$

Also, we have

$$s_1 \prec_{\oplus} s_2 \Rightarrow \phi(s_1) < \phi(s_2)$$

After $O(D)$ steps, every node in the network has received values from all of their predecessors. If those values were badly initialized, then the received values are also possibly badly valued.

For each node u, we consider the difference between the index of its final value (since the algorithm converges to a legitimate configuration where $\texttt{OUT}_u = \widehat{\texttt{OUT}_u}$) and the index of the smallest received value which is badly initialized. The biggest possible difference is $M - m$, where M is the maximum index value of $\mathbb{S}$ and m the minimum index value of $\mathbb{S}$. This difference is called d and is $O(|\mathbb{S}|)$.

For each node u, we also consider the smallest and the greatest (in the sense of increasing) r-path mapping from u to u. Let l be the length of the smallest such r-path mapping. It increases a value index by at least l. The greatest such r-path mapping increases a value index by at most d, and is of length at most d.

In the worst case, there exists a node that has an incorrect input value indexed with m, a correct input value indexed with M, so it has to wait until the incorrect value index is increased by $M-m$ before the incorrect value effect is canceled. Each l time units at least, this incorrect value index is increased by l. Again, in the worst case, if $\lfloor \frac{d}{l} \rfloor < \frac{d}{l}$, another incorrect value may still be lower than the correct value, and the greatest cycle may be followed, inducing an extra d time delay. Overall, after the first $O(D)$ time units, $\left(\lfloor \frac{d}{l} \rfloor \times l\right) + d = O(d)$ time units are needed. □

5 Application to *ad hoc* and Sensor Wireless Networks

In this section, we describe how the loose requirements of our scheme make it suitable for wireless networks such as *ad hoc* and sensor networks.

In such networks, communications are typically not bidirectionnal due to the various possible ranges of antennas, and the fact that nodes could be deployed in various geographical settings. Moreover, due to the possible collisions that can occur when neighboring nodes try to communicate at the same time, it is quite possible that messages are lost or duplicated. Also, since for example sensor networks are composed of nodes with low processing power, some desequencing is expected for message delivery when nodes are overloaded.

While some previous works on self-stabilizing sensor networks expect nodes to be aware of their location [15] or the identity of the nodes in their vicinity, the bootstrapping process that is needed to collect this information can be costly. Also, the hypothesis that nodes have unique IDs (that is mandatory to properly construct the set of identifiers in one vicinity, *e.g.* in [13]) could be falsified if such a property can not be guaranteed in practise (and larger scale construction of sensor networks would probably lead to such possibilities). Previous approaches mentioned in the introduction [1, 4, 5, 8, 11, 10] rely heavily on some kind of local knowledge about the topology: number of distinct input links, number of distinct output links, diameter of the network (for some).

In contrast, the correctness of the scheme presented in this paper does not rely necessarily on *e.g.* disctinct *neighbors*, but rather on the number of distinct *input values*. As such, our algorithm for unreliable message passing networks can be derived into a scheme for wireless networks where nodes use a local broadcast primitive to communicate with neighbors. However, only strictly idempotent r-operators that share a unique r-function are solely input value based. One such qualifying r-operator is the minc operator.

Our algorithm can be modified as follows for input value based r-operators:

1. The `IN` fixed table is replaced by an associative memory of tuples (v, a) where $v \in \mathbb{S}$ and a is time stamp. In this associative memory, v is supposed to have been received by some (possibly anonymous) neighbor node at time a. Each time a value is received through a delivered message, the entry in the associative memory is either inserted (if the value is new) or updated with a new timestamp. To prevent from bad initialization, each time the associative memory is updated, old entries are removed (following *e.g.* the technique provided in [13]).

2. Instead of computing using the `IN` table, the r-operator operates on the associative memory values.

3. Instead of sending a message to each outgoing link, nodes simply perform a local broadcast of their value.

Of course, this scheme assumes that nodes are endowed with a local real time clock (with no assumptions made about clock synchronization or possible drift), and that the timeouts are properly set so that an actual value at some node is regularly sent to the outgoing neighbors (by a local broadcast), so that those nodes in turn do not remove this value from their associative memory. Most schemes envisioned today for wireless communication between neighboring nodes are probabilistic and guarantee that between any two successful sendings, a constant amount of time is expected, provided that the density in each vicinity is upper bounded by a constant, so our hypothesis remains reasonable. Actualy tuning the timeout so that incorrect entries are quickly removed yet correct entries remain in the system is beyond the scope of this paper.

6 Concluding remarks

We presented a generic distributed algorithm for message passing networks applicable to any directed graph topology. This algorithm tolerates transient faults that corrupt the processors and communication links memory as well as intermittent faults (fair loss, reorder, finite duplication of messages) on communication media. Our contribution allows to envisage new applications for wireless networks (such as sensor networks), where nodes are not aware of their neighbors, and communications could be unidirectional (*e.g.*, non uniform power) and unreliable.

We provided evidence that our scheme is also suitable (for a restricted set of operators) to wireless networks, sur as *ad hoc* and sensor networks. Because our approach is essentially value based, computations can be carried out in potentially anonymous networks without the need of a bootstrapping process.

As an illustration, we quickly presented a simple application of the minc r-operator for solving the shortest path tree problem. Thanks to our generic approach, many others applications can be solved in the same way, by simply changing the operator. Moreover only local conditions have to be checked to insure the self-stabilization of our algorithm. Some r-operators have already been proposed for solving both fundamental and high level applications (see [10, 11]) such as: shortest paths spanning tree and related problems, best reliable paths from some transmitters, depth first search tree... More complex applications can be solved with specific r-operators, though the completeness of r-operators is an open problem.

Acknowledgements This work was supported in part by the FRAGILE and SR2I projects of the ACI "Sécurité et Informatique".

References

[1] Y. Afek and A. Bremler. Self-stabilizing unidirectional network algorithms by power supply. *Chicago Journal of Theoretical Computer Science*, 4(3):1–48, 1998.

[2] Y. Afek and G.M. Brown. Self-stabilization over unreliable communication media. *Distributed Computing*, 7:27–34, 1993.

[3] F. Baccelli, G. Cohen, G. Olsder, and J.-P. Quadrat. *Synchronization and Linearity, an algebra for discrete event systems*. Wiley, Chichester, UK, 1992.

[4] J.A. Cobb and M.G. Gouda. Stabilization of routing in directed networks. In *Proceedings of the Fifth Internationa Workshop on Self-stabilizing Systems (WSS'01), Lisbon, Portugal*, pages 51–66, 2001.

[5] S. Delaët and S. Tixeuil. Tolerating transient and intermittent failures. *Journal of Parallel and Distributed Computing*, 62(5):961–981, 2002.

[6] S. Dolev. *Self-stabilization*. The MIT Press, 2000.

[7] S. Dolev, M.G. Gouda, and M. Schneider. Memory requirements for silent stabilization. *Acta Informatica*, 36(6):447–462, 1999.

[8] S. Dolev and E. Schiller. Self-stabilizing group communication in directed networks. *Acta Inf.*, 40(9):609–636, 2004.

[9] B. Ducourthial. New operators for computing with associative nets. In *Proceedings of SIROCCO'98, Amalfi, Italia*, 1998.

[10] B. Ducourthial and S. Tixeuil. Self-stabilization with r-operators. *Distributed Computing*, 14(3):147–162, 2001.

[11] B. Ducourthial and S. Tixeuil. Self-stabilization with path algebra. *Theoretical Computer Science*, 293(1):219–236, 2003.

[12] Bertrand Ducourthial and Sébastien Tixeuil. Adaptive multi-sourced multicast. In *Rencontres Francophones sur les aspects Algorithmiques des Télécommunications (AlgoTel'2001)*, pages 135–142, St-Jean de Luz, France, May 2001. in French.

[13] Ted Herman and Sébastien Tixeuil. A distributed tdma slot assignment algorithm for wireless sensor networks. In *Proceedings of the First Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors'2004)*, number 3121 in Lecture Notes in Computer Science, pages 45–58, Turku, Finland, July 2004. Springer-Verlag.

[14] S. Katz and K.J. Perry. Message passing extensions for self-stabilizing systems. *Distributed Computing*, 7(1):17–26, 1993.

[15] Sandeep S. Kulkarni and Umamaheswaran Arumugam. Collision-free communication in sensor networks. In Shing-Tsaan Huang and Ted Herman, editors, *Self-Stabilizing Systems, 6th International Symposium, SSS 2003, San Francisco, CA, USA, June 24-25, 2003, Proceedings*, volume 2704 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2003.

[16] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.