Providing Collision-Free and Conflict-Free Communication in General Synchronous Broadcast/Receive Networks

Abdelmadjid Bouabdallah, Hicham Lakhlef, Michel Raynal, François Taïani

Providing Collision-free and Conflict-free Communication in General Synchronous Broadcast/Receive Networks

Abdelmadjid Bouabdallah † Hicham Lakhlef† Michel Raynal‡,⋆ François Taïani‡

† Sorbonne universits, Universit de technologie de Compigne, CNRS,
Heudiasyc UMR 7253, CS 60319 ; 60203 Compigne Cedex
‡ IRISA, Université de Rennes, France
⋆ Institut Universitaire de France
bouabdal@utc.fr hlakhlef@utc.fr raynal@irisa.fr francois.taiani@irisa.fr

***Abstract*—This work considers the problem of communication in dense and large scale wireless networks composed of resource-limited nodes. In this kind of networks, a massive amount of data is becoming increasingly available, and consequently implementing protocols achieving error-free communication channels constitutes an important challenge. Indeed, in this kind of networks, the prevention of message conflicts and message collisions is a crucial issue. In terms of graph theory, solving this issue amounts to solve the distance-2 coloring problem in an arbitrary graph. The paper presents a distributed algorithm providing the processes with such a coloring. This algorithm is itself collision-free and conflict-free. It is particularly suited to wireless networks composed of nodes with communication or local memory constraints.**

Keywords: Arbitrary networks, Broadcast/receive, Collision, Conflict, Distance-2 graph coloring, Message-passing, Network traversal, Synchronous system, Time slot assignment, Wireless network.

I. Introduction

a) Synchronous broadcast/receive systems: In a synchronous system, the processes execute a sequence of rounds, where a round is a bounded time slot such that any message send at the beginning of a slot is received by the end of the same slot by its receiver [15]. In such a system, a collision occurs when a process receives several messages during the same round (time slot), or when two neighbor processes send message to each other during the same round.

A broadcast/receive communication system provides processes with two operations: a broadcast which allows it to send a message to all its neighbors, and a receive which allows it to receive message. This paper considers synchronous system with broadcast/receive communication.

b) Distance-2 coloring: One way to prevent collisions and conflicts in synchronous broadcast/receive systems consists in assigning a color to each neighbor in such a way that no two processes at distance ≤ 2 have different colors. Let K the maximal number of colors that are used. Let $color_i$ be the color of p_i. Once the distance-2 coloring is done, a process is allowed to broadcast a message to its neighbors only at the rounds r such that $color_i = (r \mod K)$. It is easy to see that no neighbors processes can broadcast during the same round, and no two neighbors of a process p_i can broadcast during the same round.

c) Content of the paper: Several distance-2 coloring distributed algorithms have been proposed in the literature, in a variety of different contexts. Some address synchronous send/receive message-passing systems (e.g., [1], [16]); some address shared memory systems [3], [7]; some address self-stabilizing systems (e.g., [2], [6], [8]; some targeting wireless sensor networks (e.g., [4], [5], [9], [11], [12], [13], [14]). None of them considers the pure synchronous broadcast/receive system model, which we recently addressed in [10], where is presented a distributed distance-2 coloring algorithm suited to tree networks. This algorithm is optimal in the number of colors, namely it uses $K = \Delta + 1$ colors (where Δ is the maximal degree of the communication graph –maximal number of neighbors of any vertex-).

d) Remark: This paper extends to an arbitrary graph, the algorithm presented in [10] which considers tree networks. Due to the fact that a tree has no cycles, this "extension" is far from being trivial. Nevertheless the model and the problem are the same in both cases. Hence, to have a self-contained paper, parts of Section II and Section III of the present paper are a direct copy-and-paste of the corresponding parts of [10].

This paper extends our work to an connected graph. It is made up of 6 sections. Section II presents the computation model. Section III presents the distance-2 coloring problem. Section IV presents a distributed algorithm realizing a distance-2 coloring of an arbitrary graph. This algorithm is based on a sequential network traversal [16] enriched with "backtracking" messages used to solve coloring conflicts. Section V presents an execution of the algorithm on a simple graph including a cycle. Finally, Section VI proves its correctness.

II. Synchronous Broadcast/Receive Model

a) Processes, initial knowledge, and the communication graph: The system model consists of n sequential processes denoted p_1, ..., p_n, connected by an arbitrary communication graph.

Each process p_i has an identity id_i, which is known only by itself and its neighbors (processes at distance 1 from it). The constant $neighbors_i$ is a local set, known only by p_i, including the identities of its neighbors (and only them). As noticed in the Introduction, in order for a process p_i not to confuse its neighbors, it is assumed that no two processes at distance less than or equal to 2 have distinct identities. Hence, any two processes at distance greater than 2 may have the same identity. When computing bit complexities, we will assume that any process identity is encoded in $\log_2 n$ bits.

Let Δ_i denote the degree of a process p_i (i.e. $|neighbors_i|$) and let Δ denote the maximal degree of the process graph ($\max\{\Delta_1, \cdots, \Delta_n\}$). While each process p_i knows Δ_i, no process knows Δ (a process p_x such that $\Delta_x = \Delta$ does not know that Δ_x is Δ).

When considering a process p_i, $1 \leq i \leq n$, the integer i is called its index. Indexes are not known by the processes. They are only a notation convenience used as a subscript to distinguish processes and their local variables.

b) Timing model: We assume that processing durations are equal to 0. This is justified by the following observations: (a) the duration of the local computations of a process is negligible with respect to message transfer delays, and (b) the processing duration of a message may be considered as a part of its transfer delay.

Communication is synchronous in the sense that there is an upper bound D on message transfer delays, and this bound is known by all the processes (global knowledge). From an algorithm design point of view, we consider that there is a global clock, denoted $CLOCK$, which is increased by 1, after each period of D physical time units. Each value of $CLOCK$ defines what is usually called a *time slot* or a *round*.

c) Communication operations: The processes are provided with two operations denoted broadcast() and receive(). A process p_i invokes broadcast TAG(m) to send the message m, whose type is TAG, to all its neighbors. It is assumed that a process invokes broadcast() only at a beginning of a time slot. When a message TAG(m) arrives at a process p_i, this process is immediately warned of it, which triggers the execution of operation receive() to obtain the message. Hence, a message is always received and processed during the time slot –round– in which it was broadcast.

From a linguistic point of of view, we use the two following **when** notations when writing algorithms, where predicate is a predicate involving $CLOCK$ and possibly local variables of the concerned process.

when TAG(m) **is received do** processing of the message.
when predicate **do** code entailing one broadcast() invocation.

d) Message collision and message conflict: Traditional wired round-based synchronous systems assume a dedicated communication medium for each pair of processes (i.e., this medium is not accessible to the other processes). Hence, in these systems a process p_i obeys the following sequential pattern during each round: (a) first p_i sends a message to all or a subset of its neighbors, (b) then p_i receives the messages sent to it by its neighbors during the current round, and (c) finally executes a local computation which depends on its local state at the beginning of the round and the messages it has received during the current round.

The situation is different in systems such as wireless networks (e.g., sensor networks), which lack a dedicated communication medium per pair of processes. A process p_i shares a single communication medium with all its neighbors, and "message clash" problems can occur, each message corrupting the other ones, and being corrupted by them. Consider a process p_i, these problems are the following.

- If two neighbors of p_i invoke the operation broadcast() during the same time slot (round), a message *collision* occurs.
- If p_i and one of its neighbors invoke broadcast() during the same time slot (round), a message *conflict* occurs.

As already indicated, this paper considers this broadcast/receive communication model. This implies that protocols must prevent collisions and conflicts to ensure both message consistency and computation progress.

III. The Distance-2 Coloring Problem

a) Solving the collision/conflict problem: To prevent collisions and conflicts involving a process p_i, only a single process in the set $\{p_i\} \cup neighbors_i$ can obtain the right to communicate during a given round. To this end, we associate each process with time slots (rounds) in which it can broadcast a message, while none of its 2-hop neighbors can broadcast during these time slots. When considering the whole set of processes, this assignment must be optimal in terms of numbers of colors (ideally allowing as many processes as possible to broadcast during the same round).

This problem is a well-known graph coloring problem called *distance-2* coloring. The aim is to design distributed algorithms associating a color with each process (which will define the time-slots during which it will be allowed to broadcast) such that the following properties are satisfied.

b) Definition:

- Validity: The final color of each process belongs to $\{0, ..., K\}$, where $K \leq \Delta^2$.
- Consistency: No two processes at distance ≤ 2 have the same color at the end of the coloring algorithm.
- Termination: Each process obtains a color and one process knows that this occurred.

c) Using the colors to define the time slots: The colors obtained by the processes are used as follows, where $color_i$ is the color obtained by process p_i. The time slots (rounds) during which p_i is allowed to broadcast a message to its neighbors correspond to the values of $CLOCK$ such that $\big((CLOCK \bmod (K+1)\big) = color_i)$. As we will see, these time slots are different from the time slots used during

the (sequential and parallel) distributed distance-2 algorithms which are presented below. It follows that these algorithms must provide each process with the (initially unknown) value of K.

IV. Sequential Distance-2 Coloring of a graph

This section presents a distributed distance-2 coloring distributed algorithm for any connected graph. This solution is shown in Algorithm 1. During the execution of this algorithm there are neither message collisions, nor message conflicts. This algorithm is sequential in the sense that its skeleton is a depth-first tree traversal in which the control flow (implemented by appropriate messages) moves sequentially from a process to another one.

Algorithm 1 assumes that a single process receives a message, START(), which defines it as the root of the network. This external message causes the receiving process, p_r, to simulate the reception of a fictitious message, COLOR$(id_i, id_i, -1, 0, \emptyset)$. This message initiates a depth-first traversal of the network.

a) Messages: The algorithm uses five types of messages: COLOR(), CORRECT(), CORRECTED_COLOR(), RESUME_COLORING() and TERM(). These messages do not have the same number of parameters. As each message is received by all the neighbors of thge sender, its first parameter is the identity of its destination neighbor process. However, even if the message is not destined to it, the process can do local actions (line 10).

These messages implement a depth-first traversal of the network [16]. Each carries the identity of its destination $(dest)$, the identity of its sender $(sender)$. In addition, the message COLOR() carries the color of its sender $(sender\_cl)$, the proposed color $(proposed\_cl)$ and the color set of the already colored neighbors of the sender $(d1colors)$. The message CORRECT() carries the color of the sender and the set $d1colors$. The message CORRECTED_COLOR() carries a second identity of destination $dest2$ and the corrected color $(corrected\_cl)$. The message RESUME_COLORING() carries destination identity $dest$ and the sender identity $sender$.

Let us call the messages CORRECT(), CORRECTED_COLOR() and RESUME_COLORING() as *backtracking messages*.

b) Local variables: Each process p_i manages the following local variables.

- $state_i$ (initialized to 0) is used by p_i to manage the progress of the network traversal. Each process may traverse seven different states during the execution of the algorithm. States $1, 2, 3, 4, 5$, and 6 are active: a process in state 1 broadcasts a COLOR() message destined to one of its uncolored children, a process in state 2 broadcasts a CORRECT() message destined to its parent after the detection of a temporary inconsistency in the coloring, a process in state 3 broadcasts a TERM() message destined to its parent, a process in state 4 or 6 broadcasts a CORRECTED_COLOR() message, and a process in state 5 broadcasts a RESUME_COLORING() message to its child. States 0 is a waiting state. Nodes listen on the broadcast channel but cannot send any message.
- $parent_i$ saves the identity of the process p_j from which p_i received the message COLOR$(id_i, -, -, -, -)$; p_i receives exactly one such message. This process, p_j, defines the parent of p_i in the built tree. The root p_r of the built tree, defined by the reception of the external message START(), is the only process such that $parent_r = id_r$.
- $sender\_cl_i$ records the color of the parent of p_i. p_i receives this information in the parent's COLOR message.
- $d1colors_i$ is a set containing the colors of the neighbors of p_i, that have already obtained their color.
- $d2colors_i$ is a set containing the colors of the neighbors of neighbors of p_i (processes at distance 2), that have already obtained their color.
- to_color_i (initialized to $neighbors_i$) is a set containing the identities of the neighbors of p_i not yet colored.
- $color_i$ contains the color of p_i.

c) Description of the algorithm: Processes start the algorithm in state 0, waiting for a COLOR$(id_i, -, -, -, \emptyset)$ message, where the sender (color proposer) proposes a color to one of its neighbors. A process, p_i, receives such a message destined to it (i.e $dest = id_i$) exactly once. When it receives this message, it is visited for the first time by the depth-first tree traversal. It consequently assigns the values of the message parameters to its local variables $parent_i$, $sender\_cl_i$, $d1colors_i$ and $d2colors_i$ (lines 02, 06, 07). If the proposed color is not used by the neighbors and the neighbors of neighbors, and the color of the color proposer is not used by any process in $neighbors_i$, then the process accepts the proposed color (line 06) and moves to the state 1. So it can broadcast a COLOR() message at line 13 if it has at least one uncolored neighbor. This message is destined to this uncolored neighbor (line 12). At line 07, if a process, p_i, does not have neighbors to color, it obtains $state_i = 3$. This state allows the process to broadcast in the next round a TERM() message destined to its parent. And if all the children of a parent are colored (line 24), this parent obtains $state_i = 3$ which permits to it the broadcast of TERM() message destined to its parent and so on following lines 18, 25. Otherwise (there is an uncolored neighbor), the parent obtains the state 2 which allows, in the next round, the broadcast of a COLOR() message destined to one of its uncolored neighbors.

If a destination process, p_i, detects that the color of the color proposer is used by one of its neighbors, or the proposed color is used by a neighbor or a process at distance 2, then it does not take the proposed color, and moves to $state_i = 2$ (lines 03 and 04). This state allows it to inform the color proposer about this temporary inconsistency using a CORRECT() message at line 11. At the reception of this message, the process updates its sets $d1colors_i$ and $d2colors_i$ and it knows now all current colors of the processes at distance 2 (lines 29). At line 30 it then obtains a new color which is not used by its neighbors and the processes at distance 2. It also obtains $state_i = 4$ which permits it to broadcast a CORRECTED_COLOR() message to its neighbors (line 19). At the reception of this message the

init: $d1colors_i \leftarrow \emptyset$; $sender_i \leftarrow 0$; $d2colors_i \leftarrow \emptyset$; $state_i \leftarrow 0$; $to\_color_i \leftarrow neighbors_i$.

when START() **is received do**
(01) the reception of this message defines its receiver p_i as the root of the tree;
this process p_i simulates then the sending of COLOR$(id_i, id_i, -1, 0, \emptyset)$ to itself.

when COLOR$(dest, sender, sender_cl, proposed_color, d1colors)$ **is received do**
(02) $to\_color_i \leftarrow to\_color_i \setminus \{sender\}$; $d2colors_i \leftarrow d2colors_i \cup d1colors$;
(03) **if** $(dest = id_i) \wedge ((sender\_cl \in d1colors_i) \vee (proposed\_color \in (d1colors_i \cup d2colors_i)))$
(04) **then** $state_i \leftarrow 2$; $sender_i \leftarrow sender$;
(05) **else if** $(dest = id_i) \wedge (to\_color_i \neq \emptyset)$
(06) **then** $state_i \leftarrow 1$; $color_i \leftarrow proposed\_color$; $parent_i \leftarrow sender$; $d1colors_i \leftarrow d1colors_i \cup \{sender\_cl\}$;
(07) **else if** $(dest = id_i) \wedge (to\_color_i = \emptyset)$ **then** $state_i \leftarrow 3$; $parent_i \leftarrow sender$; **end if**
(08) **end if**
(09) **end if**;
(10) **if** $(dest \neq id_i)$ **then** $d2colors_i \leftarrow d2colors_i \cup \{proposed\_color\}$; $d1colors_i \leftarrow d1colors_i \cup \{sender\_cl\}$ **end if**.

when ($CLOCK$ increases)
(11) **if** $(state_i = 1)$ **then** $color\_for\_child \leftarrow$ the first color in $0, 1..$ which is not in $d1colors_i \cup color_i$;
(12) $next \leftarrow$ any $id_k \in to\_color_i$;
(13) broadcast COLOR$(next, id_i, color_i, color_for_child, d1colors_i)$; $state_i \leftarrow 0$
(14) **end if**;
(15) **if** $(state_i = 2)$ **then** $color_i \leftarrow$ the first color in $0, 1...$ which is not in $d1colors_i \cup d2colors_i$;
(16) broadcast CORRECT$(sender_i, id_i, color_i, d1colors_i)$; $state_i \leftarrow 0$
(17) **end if**;
(18) **if** $(state_i = 3)$ **then** broadcast TERM$(parent_i, color_i, id_i)$; $state_i \leftarrow 0$ **end if**;
(19) **if** $(state_i = 4)$ **then** broadcast CORRECTED_COLOR $(sender_i, parent_i, id_i, color_i)$; $state_i \leftarrow 0$ **end if**;
(20) **if** $(state_i = 5)$ **then** broadcast RESUME_COLORING $(sender_i, id_i)$; $state_i \leftarrow 0$ **end if**;
(21) **if** $(state_i = 6)$ **then** broadcast CORRECTED_COLOR $(-1, -1, id_i, corrected\_cl_i)$; $state_i \leftarrow 0$ **end if**.

when TERM$(dest, sender, color)$ **is received do**
(22) **if** $(dest \neq id_i)$ **then** discard the message (do not execute lines 22-27) **end if**;
(23) $to\_color_i \leftarrow to\_color_i \setminus \{sender\}$;
(24) **if** $(to\_color_i = \emptyset)$ % the neighbors of p_i are properly colored %
(25) **then if** $(parent_i = id_i)$ **then** the root claims the graph is colored **else** $state_i \leftarrow 3$ **end if**
(26) **else** $d1colors_i \leftarrow d1colors_i \cup \{color\}$; $state_i \leftarrow 1$
(27) **end if**.

when CORRECT$(dest, sender, color, d1colors)$ **is received do**
(28) **if** $(dest \neq id_i)$ **then** discard the message (do not execute lines 29-30) **end if.**
(29) $d2colors_i \leftarrow d2colors_i \cup d1colors$; $d1colors_i \leftarrow d1colors_i \cup color$;
(30) $color_i \leftarrow$ the first color in $0, 1..$ which is not in $d1colors_i \cup d2colors_i$; $sender_i \leftarrow sender$; $state_i \leftarrow 4$.

when CORRECTED_COLOR$(dest1, dest2, sender, color)$ **is received do**
(31) **if** $(dest1 \neq id_i) \wedge (dest1 \neq -1)$ **then** delete the last color added to $d1colors_i$; $d1colors_i \leftarrow d1colors_i \cup \{color\}$ **end if**;
(32) **if** $(dest1 \neq id_i) \wedge (dest1 = -1)$ **then** delete the last color added to $d2colors_i$; $d2colors_i \leftarrow d2colors_i \cup \{color\}$ **end if**;
(33) **if** $(dest2 = id_i)$ **then** $state_i \leftarrow 6$; $corrected\_cl_i \leftarrow color$ **end if**;
(34) **if** $(dest1 = -1) \wedge (dest2 = -1) \wedge (color_i = color)$ **then** $state_i \leftarrow 5$ **end if**.

when RESUME_COLORING$(dest, sender)$ **is received do**
(35) **if** $(dest \neq id_i)$ **then** discard the message (do not execute lines 36-37) **end if**;
(36) $parent_i \leftarrow sender$;
(37) **if** $(to\_color_i \neq \emptyset)$ **then** $state_i \leftarrow 1$ **else** $state_i \leftarrow 3$ **end if**.

Algorithm 1: Sequential distance-2 coloring for an arbitrary graph (code for p_i)

neighbors delete the previous colors added in the previous round to the lists $d1colors_i$ and $d2colors_i$, and replace them with the corrected colors (lines 31 and 32).

When the message CORRECTED_COLOR() is received by the parent of the process that detected the temporary inconsistency it obtains the state 3 in line 34. This state allows to this process the broadcast of RESUME_COLORING() message destined to its child that sent the message CORRECT() (line 20). At the reception of this message, the process obtains the state 2 if it has neighbors to color, else it obtains the state 3 (signaling local termination) which allows the broadcast of a TERM() message destined to its parent.

V. A Simple Example

Considering the network of Figure 1, an example of execution of Algorithm 1 is depicted in Table I. Due to space restriction we abbreviate the following:

- br = broadcast operation,
- $d1_i$ = $d1colors_i$,
- $d2_i$ = $d2colors_i$,
- CL(p_a,p_b,c,z,S) = COLOR (p_a,p_b,c,z,S),
- CRL(p_a,p_b,c,S) = CORRECT(p_a,p_b,c,z,S),
- CR_CL(p_a,p_b,c,S) = CORRECTED_CL(p_a,p_b,c,S),
- RSM_CL(p_a,p_b) = RESUME_CL(p_a,p_b),

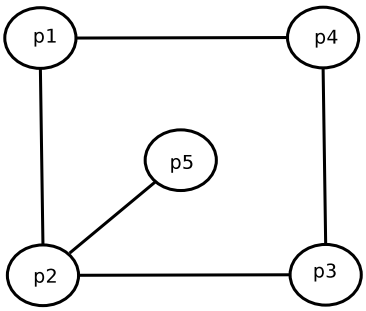

Figure 1. A 5-process arbitrary network

- $s_i = state_i$.

Process p_1 receives the message START() at round -1. It broadcasts the message COLOR (id_1, id_1, -1, 0, $\emptyset$) to itself. It receives this message at round 0. It updates its local variables $d2_1$ at line 02 (abbreviated as L2), $d1_1$, s_1 and $color_1$ at line 06 (abbreviated as L6). This appears in the first row of the table where the value of $CLOCK$ is 0. Then, when $CLOCK$ progresses to 1, p_1 has $state_1 = 1$, therefore, it broadcasts the message COLOR() with appropriate parameters, where it proposes a color that is not in $d2_1 \cup d1_1$ to each of its neighbors, and subsequently enters $state_1 = 0$ (L13). This appears in the second row of the table where the value of $CLOCK$ is 1. When p_2 and p_4 receive this message at round 2, they execute the associated processing at L2 and L6 for p_2 and at L10 for p_4. So, in this round p_2 updates its local variables $d2_2$ at L2, $d1_2$, s_2 and $color_2$ at L6. p_4 updates its local variables $d2_4$ at L2, $d1_4$, at L10. This appears in the third row of the table where $CLOCK$ is 2. In round 3, p_2 has $state_2 = 1$, so it broadcasts the message COLOR() with appropriate parameters (L13) and obtains $state_2 = 0$ (L13), Etc.

In round 6, p_4 finds that the color proposed by its neighbor p_3 (color 0 proposed in round 5) is in $d1_4$, so it "refuses" this color and obtains $state_4 = 2$ (L4). In round 7, p_4 obtains a color which is not in $d2_4 \cup d1_4$ (L15) and broadcasts the message CORRECT(id_3, id_4, 2, 0, $\{0\}$). In round 8, p_3 updates $d1_3$ and $d2_3$ and obtains a new color (L30). Its state will allow it to broadcast the message CORRECTED_CL() (round 11). The broadcast of this message will trigger the broadcast of the message RESUME_CL() (round 15) to resume the coloring as described before.

VI. Proof and Cost of the Algorithm

Lemma 1. *Algorithm 1 is collision-free and conflict free.*

Proof Let us recall that, in state 0, a process cannot broadcast messages. The states in $S = \{1, 2, 3, 4, 5, 6\}$ are active states. Let us observe that after the reception of the message START(), there is only one process (the initiator) that can broadcast a message to its neighbors in the next round (it enters state 1, line 06). The other processes are initially in state 0, which prevents them from broadcasting a message.

Let us now observe that, after the reception of a message m, a process p_i can change its state in two cases:

- Case 1. The message m is for it ($dest = id_i$), or
- Case 2. m = CORRECTED_COLOR$(-1, -1, -, color)$, where $color_i = color$.

In the first case the sender process puts only one identity destination in each message, due to lines 04, 06 and 12. Let us observe that p_i cannot move to an active state if it is not the destination of message sent from one of its neighbors, p_j (lines 03, 05, 07, 22, 28, 31, 35). The broadcasting of this message by p_j entails the immediate assignment $state_j \leftarrow 0$. It follows that the sender process and the destination process cannot broadcast in the same round. Each time a message is broadcast by a process p_j, there is only one neighbor of p_j that will move to an active state. Therefore, at each round there is no two processes p_i and p_q in $neighbors_j$ that have $state_i$ and $state_q$ in S. It follows that there will be only one neighbor of p_j authorized to broadcast a message in the next round.

In the second case we have to prove that there is only one process that moves to the state 5 after the broadcast of the message CORRECTED_COLOR$(-1, -1, -, color)$. Let p_f be the the sender of this message. Since p_f is broadcasting this message, it has the state 6. It got this state in the previous round because it has received message CORRECTED_COLOR$(-, id_f, -, -)$ from its child p_z that has $color_z = color$. Let us assume that there is another process $p_x \in neighbor_f$ that has $color_x = color$. If p_x is the process that broadcast a COLOR() message destined to p_f (p_x colored p_f), then due to line 11 p_f does not propose the color $color$ to its children. And if p_f is the process that broadcast a COLOR() message to p_x, then p_z is aware about that (line 10) and know that the color $color$ is used by a process at distance 2. Therefore, p_z cannot accept this color and moves to the state 2 to correct the proposition. It follows that the message CORRECTED_COLOR$(-1, -1, -, color)$ is destined to only one neighbor.

Hence, the control flow generated by these messages remains sequential, moving sequentially from a parent process to a child process for the messages COLOR(), CORRECTED_COLOR() and RESUME_COLORING(). Similarly, the control flow generated by the messages TERM() and CORRECT() moves sequentially from a child process p_i to its parent process (whose identity is saved in $parent_i$).

The collision-freedom and conflict-freedom properties of the algorithm follow directly from the sequentiality of the control flow realized by the messages. $\square_{Lemma\ 1}$

Lemma 2. *No two processes at distance ≤ 2 obtain the same color.*

Proof Let us observe that the initiator takes the first color 0 (line 01) and each process may propose in a COLOR() message one color to only one of its uncolored neighbors (lines 11-14).

Let us recall that each process in the network receives messages only from its one hop neighbors. Therefore, only neighbors know if a given color is proposed to neighbors of neighbors (line 10). Due to the instruction in line 11, a process

p_i / clock	p_1	p_2	p_3	p_4	p_5
0	$d2_1 = \{\}$ (L2),$d1_1 = \{-1\}$ $s_1 = 1$, $color_1 = 0$ (L6)				
1	br CL(id_2,id_1,0,1,$\{-1\}$) $s_1 = 0$ (L13)				
2		$d2_2 = \{-1\}$ (L2),$d1_2 = \{0\}$ $s_2 = 1$, $color_2 = 1$ (L6)		$d2_4 = \{1\}$,$d1_4 = \{0\}$ (L10)	
3		br CL(id_3,id_2, 1, 2, $\{0\}$) $s_2 = 0$ (L13)			
4	$d2_1 = \{-1, 2\}$, $d1_1 = \{1\}$ (L10)		$d2_= \{0\}$ (L2),$d1_3 = \{1\}$ $s_3 = 1$, $color_3 = 2$ (L6)		$d2_5 = \{0, 2\}$ $d1_5 = \{1\}$ (L10)
5			br CL(id_4,id_3, 2, 0, $\{1\}$) $s_3 = 0$ (L13)		
6		$d2_2 = \{-1, 0\}$, $d1_2 = \{0, 2\}$ (L10)		$d2_4 = \{1\}$ (L2), $s_4 = 2$ (L4)	
7				$color_4 = 2$ (L15), br CR(id_3,id_4,2,$\{0\}$) $s_4 = 0$ (L16)	
8			$d2_3 = \{0\}$, $d1_3 = \{1, 2\}$ (L30) $color_3 = 3$, $s_3 = 4$ (L31)		
9			br CR_CL(id_4,id_2,id_3,3) $s_3 = 0$ (L19)		
10		$d1_2 = \{0, 3\}$ (L31), $s_2 = 6$ (L33)			
11		br CR_CL(-1,-1,p_2,3) $s_2 = 0$ (L21)			
12	$d2_1 = \{-1, 3\}$ (L32)				
13			$s_3 = 5$ (L34)		$d2_5 = \{0, 3\}$ (L32)
14			br RSM_CL(id_4, id_3) $s_3 = 0$ (L20)		
15				$s_4 = 3$ (L37)	
16				br TERM(id_3, id_4) $s_4 = 0$ (L18)	
17			$s_3 = 3$ (L37)		
18			br TERM(id_2, id_3) $s_3 = 0$ (L18)		
19		$s_2 = 1$ (L26) $d1_2 = \{0, 3\}$ (L26)			
20		br CL(id_5,id_2,1,2,$\{0, 3\}$) (L16) $s_2 = 0$ (L13)			
21	$d2_1 = \{-1, 3, 2\}$, $d1_1 = \{0, 1\}$ (L10)		$d2_3 = \{0, 2\}$, $d1_3 = \{2, 1\}$ (L10)		$d2_5 = \{0, 3\}$ $s_5 = 3$ (L07)
22					br TERM(id_2,id_5) $s_5 = 0$ (L18)
23		$s_2 = 3$ (L25)			
24		br TERM(id_1, id_2) $s_2 = 0$ (L18)			
25	end algorithm (L25)				

Table I
AN EXECUTION OF ALGORITHM 1 ON THE NETWORK OF FIGURE 1

proposes to its neighbors colors which are different from its color and different from the colors of its colored neighbors.

Although each process, p_j, does not propose $color_j$ and $d1colors_j$, a colored process p_i at distance 2 from a colored process p_j may propose the same color of p_j to one of its uncolored neighbors p_e which is at distance 2 from p_j. Which violates the consistency propriety. Also, a non-colored process p_e that has at least two colored neighbor p_i and p_j may receive as proposed color the color of p_j from p_i if p_i is not p_j's neighbor. Which creates two neighbors with the same color and violates the consistency propriety. This cases may occur because the process does not see the process at distance 1 or 2 of its neighbors.

However, due to line 03 any violation of this kind is detected. Because the color of p_j is already stored in $d1colors_i$ and the colors of the neighbors of p_j are already stored in $d2colors_i$ (line 10). If the test of the instruction in line 03 is true for p_i, p_i will not be able to propose a color (broadcast a color message) because its state now is $state_i = 2$. The possession of this state entails the broadcast of a CORRECT() message to correct the *temporary inconsistency* by p_e that obtains a new color which is different from the colors of its neighbors and the neighbors of its neighbors (line 15).

Due to the instruction in (line 30), the process correct its color based on the information sent by the process that detected the inconsistency and moved to $state_i = 4$. This state permits to p_e to broadcast a CORRECTED_COLOR() message to inform its neighbors about the corrected colors (line 19). Due to the instruction in (line 33), the broadcast of the CORRECTED_COLOR() message entails the the broadcast

of another CORRECTED_COLOR message by one neighbor (the parent) of p_e to inform its neighbor about the new color of its child. And this broadcast entails the broadcast of RESUME_COLORING() message destined to the process p_i that detected the inconsistency. Therefore, the process p_i that detected the inconsistency cannot take the $state_i = 1$ ((line 37) until all process which are at distance 2 for the corrected process are have received the message CORRECTED_COLOR(). It follows that before p_i resumes the coloring, the current network is colored partially correctly. These complete the proof the lemma. $\square_{Lemma\ 2}$

Lemma 3. *Any process is colored, and (only after all processes are colored) this is known by the root process.*

Proof If the network is such that there is no temporary inconsistency: in this case due to the instruction in line 06, after the reception of the message color, the process broadcasts in the next round a color message destined to one of its uncolored neighbors. Due to lines 06 and 24, a process does not broadcast a TERM() message if it has at least one uncolored neighbor because it cant not move to the state 3. It follows that before the root receive all TERM() messages from all its neighbors each process receives a message color.

If the network is such that there is temporary inconsistency: in this case let us observe that the process p_i that detected the temporary inconsistency does not broadcast a COLOR() message in the next round because it does not have the state 1 (line 04). Following line 16, it broadcasts a CORRECT() message destined to its parent p_z. Due to line 33 and line 21 p_z receives the message corrected_color$(-1, -1, -, color)$ broadcast by its parent (as discussed in the proof of lemma 1) which allows the broadcast of the message RESUME_COLORING() destined to its child p_i. Then, at the reception of this message,p_i moves to the state 1. Therefore, p_i resume, in a finite time, the coloring at line 11.

Due to line 25 each process that have all neighbors colored broadcast a TERM() message destined to its parent.

It follows from these observations that each process eventually obtains one color and the root receives a term message from each neighbors. $\square_{Lemma\ 3}$

The following theorem is an immediate consequence of the previous lemmas.

Theorem 1. *Algorithm* 1 *is a collision-free and conflict-free distance-2 coloring algorithm for any connected graph.*

a) Cost of the algorithm: (Let us recall that a process identity can be encoded with $O(\log n)$ bits.) There are five message types. A message TERM() carries two process identities and a color. A message COLOR() carries two process identities, two colors, and set of at most Δ colors. A message CORRECT() carries two process identities, a color, and set of at most Δ colors. A message CORRECTED_COLOR() carries three process identities, and two colors. A message RESUME_COLORING() carries two process identities. It follows that a message carries at most $9 \log n + (2\Delta + 6) \log \Delta$ bits.

Let us observe that each process in the network receive only one message color destined to it.

Let us assume an execution of the algorithm in a network where there will be not a call to broadcast the messages of backtracking (no temporary conflict of colors). Therefore, in this case there are $n - 1$ of COLOR() messages (the initiator does not receive such a message from its neighbors) and $n-1$ of TERM() messages (the last colored process in the network does not receive a TERM() message destined to it). This means there is two messages per each link of the tree built. Therefore, in this best case $2(n - 1)$ broadcasts are required.

Let us assume now that the network is such that there is temporary conflicts of colors (there are calls to the broadcast of backtracking messages.) In this case, there is, in the worst case, four more messages per link: a message CORRECT(), two messages CORRECTED_COLORING() and a message RESUME_COLORING(). Therefore, the number of broadcasts required in the worst case is bound by $6(n - 1)$.

VII. Conclusion

There is a major difficulty in designing of a efficient communication protocols when these protocols concern networks connected in an arbitrary manner without any condition of the topology nor on the size of the network, nor on the node characteristics.

An interesting open problem is the following: genalize the previous algorithm solution to a m-coloring problem, where a process can get more than one color. And how can we derive, starting from this protocol, solutions that deal with dynamic of the network, in the sense mobility and appearance and disappearance of processes.

VIII. Acknowledgments

The authors wish to express their appreciation to the anonymous reviewers for their constructive comments.

The authors wish also to express their appreciation to the Labex MS2T for its support.

References

[1] Barenboim L., Elkin M., and Kuhn F., Distributed (Delta+1)-coloring in linear (in Delta) time. *SIAM Journal of Computing*, 43(1):72-95, 2014.

[2] Blair J. and Manne F., An efficient self-stabilizing distance-2 coloring algorithm. *Proc. 16th Colloquium on Structural Information and Communication Complexity (SIROCCO'10)*, Springer LNCS 5869, pp. 237-251, 2009.

[3] Bozdag D., Gebremedhin A.S., Manne F., Boman G. and Çatalyürek U.V., A framework for scalable greedy coloring on distributed-memory parallel computers. *Journal of Parallel and Distributed Computing*, 68(4):515-535, 2008.

[4] Chipara O., Lu C., Stankovic J., and Roman. G.-C., Dynamic conflict-free transmission scheduling for sensor network queries. *IEEE Transactions on Mobile Computing*, 10(5):734-748, 2011.

[5] Ergen S.C. and Varaiya P., PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks. *IEEE Transaction on Mobile Computing*, 5(7):920-930, 2005.

[6] Gairing M., Goddard W, Hedetniemi S. T., Kristiansen P., and McRae A. A., Distance-two information in self-stabilizing algorithms. *Parallel Processing Letters*, 14(03-04):387–398, 2004.

[7] Gebremedhin A.H., Manne F., and Pothen A., Parallel distance-k coloring algorithms for numerical optimization. *Proc. Euro-Par Parallel Processing*, Springer LNCS 2400, pp. 912-921, 2002.

[8] Herman T., Tixeuil S., A distributed TDMA slot assignment algorithm for wireless sensor networks. *Proc. Int'l Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS'04)*, Springer LNCS 3121, pp. 45-58, 2004.

[9] Jemili I., Ghrab D., Belghith A., Derbel B., and Dhraief A., Collision aware coloring algorithm for wireless sensor networks. In *Proc. 9th Int' lWireless Communications and Mobile Computing Conference (IWCMC'13)*, pages 1546-1553, 2013.

[10] Frey D., Lakhlef H., and Raynal M., Optimal collision/conflict-free distance-2 coloring in wireless broadcast/receive synchronous tree networks. *Proc. 45th Annual Conference on Parallel Processing (ICPP'16)*, pp. 350-359, 2016.

[11] Mahfoudh S., Chalhoub G., Minet P., Misson M., and Amdouni I., Node coloring and color conflict detection in wireless sensor networks. *Future Internet*, 2(4):469, 2010.

[12] Prabhakar B., Uysal-Biyikoglu, E., and El Gamal A., Energy-efficient packet transmission over a wireless link. *IEEE/ACM Transactions on Networking*, 10(4):487499, 2002.

[13] Ramanathan S., A unified framework and algorithms for (T/F/C)DMA channel assignment in wireless networks. *Proc. 16th IEEE INFOCOM Conference*, IEEE Press, pp. 900-907, 1997.

[14] Lakhlef H., Bourgeois J., Harous S., Myoupo J.: Collision-Free Routing Protocol in Multi-hop Wireless Sensor Networks. *In CIT 2015, The 15th IEEE Int. Conf. on Computer and Information Technology, Liverpool, UK, pages 9299, October 2015*

[15] Raynal M., *Fault-tolerant agreement in synchronous message-passing systems.* Morgan & Claypool Publishers, 165 pages, 2010 (ISBN 978-1-60845-525-6).

[16] Raynal M., *Distributed algorithms for message-passing systems*. Springer, 500 pages, 2013, ISBN 978-3-642-38122-5.