# Gathering of Robots in a Ring with Mobile Faults

Shantanu Das[1], Riccardo Focardi[2], Flaminia L. Luccio[2],
Euripides Markou[3], Davide Moro[2], and Marco Squarcina[2]

[1] LIF, Aix-Marseille University, Marseille, France
[2] DAIS, Università Ca' Foscari Venezia, Venezia, Italy
[3] DCSBI, University of Thessaly, Lamia, Greece

**Abstract.** In this paper, we consider the problem of gathering mobile agents in a graph in the presence of mobile faults that can appear anywhere in the graph. Faults are modeled as a malicious mobile agent that attempts to block the path of the honest agents and prevents them from gathering. The problem has been previously studied by a subset of the authors for asynchronous agents in the ring and in the grid graphs. Here, we consider synchronous agents and we present new algorithms for the unoriented ring graphs that solve strictly more cases than the ones solvable with asynchronous agents. We also show that previously known solutions for asynchronous agents in the oriented ring can be improved when agents are synchronous. We finally provide a proof-of-concept implementation of the synchronous algorithms using real Lego Mindstorms EV3 robots.

## 1 Introduction

An important problem in the area of distributed computing with mobile agents (or robots) is the *rendezvous* problem, i.e., the gathering of all agents at the same location. Gathering is a crucial task for teams of autonomous mobile robots, since they may need to meet in order to share information or coordinate. This problem has been widely studied when the environment is modelled as an undirected graph and the mobile agents can move along the edges of the graph. Most of the studies are restricted to fault-free environments and very little is known about gathering in the presence of faults. Possible faults can be a permanent failure of a node, like for example the so called *black hole* that destroys agents arriving at a node, or, transient faults that can appear anywhere in the graph, such as a mobile hostile entity (an *intruder*) that behaves maliciously. Some work has been done to locate a malicious node in a graph (see, e.g., [8,10]), whereas protecting the network against a malicious entity that is mobile and moves from node to node of the graph, is a more difficult problem. An example of this kind is the so-called *network decontamination* or *intruder capture* problem (see, e.g., [9,11]).

In this paper we consider a new type of *malicious agent* that was introduced in [5]. This agent can block the movement of an honest agent to the node it occupies, it can move arbitrarily fast along the edges of the graph, it has full information about the graph and the location of the agents, and it even has full knowledge of the actions that will be taken by the honest agents. On the other hand, the honest agents are relatively weak: they are independent and identical anonymous processes with some internal memory, they move synchronously and use face to face communication when they are in the

same node. We investigate the feasibility of rendezvous in oriented and unoriented ring networks in the presence of such a powerful adversary, that blocks other agents from having access to parts of the network.

We consider an oriented and an unoriented ring network modelled as a connected undirected graph with a set of mobile honest agents located at distinct nodes of the graph, and a malicious agent that cannot harm the honest agents but can prevent them from visiting a node. As in [5] we are interested in solving the rendezvous of all honest agents in this hostile environment. However, differently from [5], we consider synchronous mobile honest agents. These agents have local communication capability, and no prior knowledge of their number $k$ (except for the protocol for $k = 2$) and of the ring size $n$. Our objective is to study the feasibility of rendezvous with minimal assumptions.

**Contributions.** $(i)$ We prove that in an unoriented ring network of $n$ nodes the problem is not solvable for $n$ odd and $k$ even, and we then present an algorithm that solves the problem in all the other cases for $k > 2$ synchronous agents, also in the case $n$ odd and $k$ odd which is unsolvable with asynchronous agents; $(ii)$ we briefly discuss how to solve the problem for $k = 2$ agents and $n$ even, not solvable in the asynchronous setting, and how to solve the problem in the oriented ring by improving the known algorithm for asynchronous agents; $(iii)$ we show that the proposed algorithms and underlying assumptions are realistic through a proof-of-concept implementation using Lego Mindstorms EV3 robots.

**Related work.** In this paper we consider the rendezvous problem, which has been previously studied for robots moving on the plane [3], or for agents moving on graphs [1]. This problem can be easily solved, in synchronous and asynchronous settings, when the system is not symmetric, e.g., when there is a distinguished node. In symmetric settings (e.g., in the anonymous ring) the problem is non-trivial and symmetry has to be broken, e.g., by using tokens [6], by adding distinct identifiers to agents [4], etc.

Some recent work [7] also considers the problem of rendezvous in the presence of *Byzantine agents*, which are indistinguishable from the *honest* agents, but may behave in an arbitrary manner, and may also provide false information to the honest agents so to prevent their gathering. Other related work includes gathering in networks with delay faults [2]. In [7] they use a model similar to ours to gather agents in the presence of an adversary that can delay for an arbitrary, but finite time, the movements of the agents. Contrary to the model of [7], in our setting the agents may be blocked forever.

From a practical point of view, the work that is more closely related to our model is [13], where the author proposes an example of a simulation of a known distributed fault-free gathering algorithm, using real NXT robots. Differently from our setting, in [13] robots are moving on a plane (not in a graph), do not use direct communication, but only indirect one, i.e., can only detect the position and the movement of the other robots but cannot communicate any information.

**Paper structure.** The paper is organized as follows: In Section 2 we illustrate the formal model; in Section 3 we first present some impossibility results, and then the algorithm for the unoriented ring network for $k > 2$ agents. For lack of space we only discuss the solution for $k = 2$ and for the oriented ring network. In Section 4 we present the implementation of the algorithms on programmable robots, and we discuss the use-

fulness of the study of the real model in the development of the theoretical models. We conclude in Section 5.

## 2 A Model of Synchronous Agents in a Ring

We consider a distributed network modeled by an undirected, connected graph $G = (V, E)$ where $V$ are the anonymous and identical nodes or *hosts*, and $E$ the edges or connections between nodes. We assume that in the network there are $k$ *honest* anonymous identical synchronous agents $A_1, A_2, \ldots A_k$, and one *malicious* agent $M$ which is trying to prevent the gathering of honest agents. More precisely, the network, and the capabilities and behavior of honest and malicious agents are the following.

**The network:** We consider an oriented and an unoriented ring network $R = (V, E)$, with $|V| = n$ nodes. The links incident to a host are distinctly labeled. In the oriented ring we assume that all agents are able to agree on a common sense of direction. In the unoriented ring this port labeling is not globally consistent. The edges of the network are FIFO links, meaning that all agents, including the malicious one, that move in the same link respect a FIFO ordering. We call a node $v$ *occupied* when one or more *honest* agents are in $v$, and we call $v$ *free* or *unoccupied* otherwise. Moreover, for solvability reasons, we will have to assume that the ring contains a special node marked $\otimes$, otherwise gathering is impossible (see [5]).

**Honest agents:** The agents are independent and identical, anonymous processes with some internal constant memory, except for the case $k = 2$ in the unoriented ring that requires $O(\log n)$ bits. These agents are initially scattered in the graph, i.e., at most one agent at a node, start the protocol at the same time, and can move along the edges of $G$. An agent located at a node $v$ can see how many other agents are at $v$, and it can access their states. An agent cannot see or communicate with any agent that is not located at the same node, i.e., communication is face to face. Moreover an agent cannot mark the node or leave any information on the node that it visits. An agent arriving at a node $v$, learns the label of the incoming port and the label of the outgoing port. Two agents traveling on the same edge in different directions do not notice each other, and cannot meet on the edge. Their goal is to rendezvous at a node. The agents neither have knowledge of the size $n$ of the ring, nor of the number $k$ of agents present in the network.

The moves of honest agents are synchronized as follows: $(a)$ all agents start executing the protocols at the same time; $(b)$ time is discretized into atomic time units; $(c)$ all agents start in the same initial state, and the set of agent states is finite and independent of the graph size or the number of agents; $(d)$ During each time unit, an agent arriving at a node $v$ through a port $q$ takes the following three actions: $(d.1)$ it reads its own state, counts the number of agents at $v$ and reads their states; $(d.2)$ based on the above information it performs some computation to decide its next destination; the output of the computation is a new state and the port number $p$ of an edge incident to $v$, or $p = 0$ if the agent decides to stay in $v$; $(d.3)$ the agent changes its state and either moves using the computed port number ($p > 0$), or waits at the current node ($p = 0$). If the agent decided to move on edge $(v, z)$, and the node $z$ is not occupied by the malicious agent then the agent is located at the node $z$ in the next time unit. Otherwise, the agent is still

located at node $v$ in the next time unit, with a flag set in its memory notifying the agent that the move was unsuccessful. In the next time unit, the agent repeats the above three actions.

**Malicious agent:** We consider a worst case scenario in which the malicious agent $M$ is a very powerful entity compared to honest agents: It has unlimited memory; at any time has full knowledge of the graph, the locations of the honest agents and their states, and the algorithm followed by the agents. Based on the above information, $M$ can either stay at its current node $y$ or decide to move to another node $w$, if there is a path from $y$ to $w$, such that no node on this path, including $w$, is occupied by any honest agent. The speed of the malicious agent is unbounded, thus $M$ can move arbitrarily fast inside the network, but must move along the edges of the graph, and it also obeys the FIFO property of the links. When it resides at a node $u$ it prevents any honest agent $A$ from visiting $u$, i.e., it "blocks" $A$: If an agent $A$ attempts to visit $u$, the agent receives a signal that $M$ is in $u$. $M$ can neither visit a node which is already occupied by some honest agent, nor cross some honest agent in a link.

## 3 Rendezvous in Rings with Synchronous Agents

In this section we first study the rendezvous problem with synchronous agents having constant memory, in an unoriented ring with one malicious agent. In [5] the authors have analyzed the same problem with asynchronous agents and have proved that the problem is solvable in any unoriented ring if and only if the number of honest agents $k$ is odd. We prove here that with $k > 2$ synchronous agents having constant memory, the problem can be solved in an unoriented ring consisting of $n$ nodes, when $n$ is even or $k$ is odd. We also prove that for $k$ even and $n$ odd numbers, the problem is unsolvable. We discuss also the case $k = 2$ in unoriented rings, and the problem in oriented rings.

Let us first show the impossibility result in the unoriented ring. The technique of the proof is similar to the one presented in Lemma 5 of [5], where it was proved that an even number of agents with constant memory cannot gather in an unoriented ring with a malicious agent. We can show that when $n$ is odd and $k$ is even, an adversary can initially place the agents in a configuration that is symmetric with respect to a line passing through the special node $\otimes$. Hence, the agents are distributed into two groups of equal size. Moreover, the agents belonging to the same group, agree on the clockwise (*CW*) orientation but they do not agree with the agents of the other group. In other words, what is considered clockwise direction for any agent of one group is considered counter-clockwise direction for every agent of the other group. Each two symmetrically placed agents which belong to different groups should behave identically under any algorithm, and therefore these two agents can never meet at a node. We thus have:

**Lemma 1 (Impossibility when $n$ is odd and $k$ is even).** *In an unoriented ring with a specially marked node $\otimes$ and a malicious agent having arbitrary speed, $k \geq 2$ mobile synchronous agents starting from arbitrary symmetric locations, cannot gather at a node if $n$ is odd and $k$ is even.*

### 3.1 Unoriented Ring with more than two Agents

We now propose an algorithm for $k > 2$ synchronous agents with constant memory, and that do not know $n$ and $k$. Moreover, agents do not agree on a common sense of direction given that the ring is unoriented. Algorithm 1, called *CollectAgents*, works as follows. Each agent has a state and some local variables that keep track of the execution. Agents wake up in state *INITIAL*. We illustrate the algorithm distinguishing the case in which all agents have the same orientation, from the case in which the agents are split in two groups, one that moves in one direction and the other in the opposite one. Agents are not aware of this information so they cannot adapt their behaviour accordingly.

In the first case if *M* tries to escape, moving in the same direction, then, the first agent that reaches $\otimes$ twice will stop and block *M*. This is important otherwise the agents will infinitely move around the ring and never rendezvous. More precisely, an *INITIAL* agent becomes *STAR* if it reaches $\otimes$ twice, i.e., when *s_reached* $= 2$ (line 15). In this case, and also if *M* independently decides to stop, another agent, let us call it $A$, will eventually arrive, and will be blocked by *M* becoming *STOPPED* (line 6). At this point it will wait for another *INITIAL* agent, let us call it $B$, that will become *MSG* (line 11). $A$ will change direction to counter-clockwise (*CCW*), going all the way around up to when it will be blocked again by $M$, becoming *R_MSG* (line 64). $B$ will then reverse direction for the last time and move towards $A$, eventually collecting an agent blocked in $\otimes$ that will become *FOLLOWER* (line 54). Note that, all the agents in the meantime have reached $A$ that has become *HEAD* after meeting $B$ (line 47), thus when $B$ reaches $A$ it rendezvous with all the agents (line 21). Notice that an agent in state *HEAD* proceeds clockwise but stops if it reaches $\otimes$ twice (line 34), in order to guarantee termination.

In the second case there are two groups of agents, G1 and G2, one that moves in one direction and the other in the opposite one. Thus, no matter how *M* moves, eventually one agent per group, i.e. $A$ for $G_1$, and $B$ for $G_2$, will be blocked by $M$ and become *STOPPED* (line 6). When other agents meet $A$ and $B$, respectively $C$ and $D$ (as we will see in an extreme case we will only have $C$ or $D$), then $A$ and $B$ will become *HEAD* and will start moving, trying to surround *M*, while $C$ and $D$ become *MSG*, reverse direction and move towards $B$ and $A$, respectively. Then, we have two subcases. If $C$ or $D$ meets a *STOPPED* agent (line 71), then it collects it, reverses direction and rendezvous at the opposite side (this is the extreme case in which $k-1$ agents move in one direction and 1 in the opposite one, and one between $A$ and $B$ is a *HEAD* agent, and the other one is *STOPPED*). In the other case they both meet the two *HEAD* agents $B$ and $A$ (line 67), they reverse direction and become *R_MSG* delivering some information and joining all the agents of their own group. So now agents of group $G_1$ (respectively, $G_2$) have gathered at the same node. The last phase consists of letting the two groups join. If $k$ is odd, either $G_1$ or $G_2$ is odd. W.l.o.g., let us assume $G_2$ is odd, then $G_2$ remains stopped (line 22) and all the agents of $G_1$ move towards $G_2$ (line 23), and gather. If $k$ is even, both $G_1$ and $G_2$ are of even size, thus the idea is to let $C$ and $D$ to go back and forth, up to when they find that *M* has been surrounded, i.e., when *HEAD* agents cannot move anymore (line 24). At this point $G_1$ and $G_2$ move towards the middle point and gather there if $n$ is even (the distance between $G_1$ and $G_2$ is $n-1$ an odd number), or failing if $n$ is odd (line 38). Finally, note that the oddness or evenness of a group is computed using only one parity bit, thus using constant memory.

**Algorithm 1** *CollectAgents*

```
    # Gathering k > 2 agents in an Unoriented Ring.
    # Local variables: sync = parity = s_reached = 0, moved = -1, first_clock = 1, State = INITIAL

 1: sync := (sync + 1) mod 2

 2: if State = INITIAL then
 3:     dir := CW
 4:     if first_clock = 1 and Reached ⊗ then s_reached = s_reached +1
 5:     end if
 6:     if Blocked then State := STOPPED
 7:     else
 8:         Move to the next node
 9:         if Reached ⊗ then s_reached = s_reached +1
10:         end if
11:         if meet a STOPPED agent then State := MSG
12:         else if meet a HEAD agent h then State := FOLLOWER of h
13:         else if Blocked then State := STOPPED
14:         else if s_reached = 2 and not found a STAR agent then
15:             State := STAR
16:         end if
17:     end if

18: else if State = HEAD then
19:     if meet a INITIAL agent then parity:= (parity + 1) mod 2
20:     else if meet a R_MSG agent r then
21:         if r.moved = -1 then State := RENDEZVOUS
22:         else if parity = 1 and r.parity = 0 then State := WAIT
23:         else if parity = 0 and r.parity = 1 then State := R_HEAD
24:         else if moved = 0 and r.moved = 0 then
25:             State := R_HEAD
26:             Wait (sync)
27:         end if
28:         moved:= 0
29:     else if not blocked and s_reached < 2 then
30:         moved:= 1
31:         Move to the next node
32:         if Reached ⊗ then s_reached = s_reached +1
33:         end if
34:     else Wait (1)
35:     end if

36: else if State = R_HEAD then
37:     dir:= CCW
38:     if Blocked then State := FAILURE
39:     else if meet a R_HEAD agent then State := RENDEZVOUS
40:     else
41:         Move to the next node
42:         if meet a R_HEAD or WAIT agent then State := RENDEZVOUS
43:         end if
44:     end if
```

*CollectAgents* algorithm - Continue

45: **else if** State = *STOPPED* **then**
46: **if** meet a *INITIAL* agent **then**
47: State := *HEAD*
48: *moved*:= 0
49: *parity*:= $(parity + 1)\ mod\ 2$
50: **else if** meet a *MSG* agent $m$ **then** State := *FOLLOWER* of $m$
51: **else** Wait (1)
52: **end if**

53: **else if** State = *STAR* **then**
54: **if** meet a *R_MSG* agent $r$ **then** State := *FOLLOWER* of $r$
55: **else** Wait (1)
56: **end if**

57: **else if** State = *FOLLOWER* **then**
58: **if** leader state is *RENDEZVOUS* **then** State := *RENDEZVOUS*
59: **else** follow the leader
60: **end if**

61: **else if** State = *MSG* **then**
62: *dir*:= *CCW*
63: **if** Blocked **then**
64: State := *R_MSG*
65: **else**
66: Move to the next node
67: **if** meet a *HEAD* agent $h$ **then**
68: state := *R_MSG*
69: *moved*:= $h$.*moved*
70: *parity*:= $h$.*parity*
71: **else if** Blocked or meet a *STOPPED* agent **then** state := *R_MSG*
72: **end if**
73: **end if**

74: **else if** State = *R_MSG* **then**
75: *dir*:= *CW*
76: Move to the next node
77: **if** meet a *HEAD* agent $h$ **then**
78: **if** *moved* = -1 **then** State := *RENDEZVOUS*
79: **else if** (*parity*!= $h$.*parity*) or (*moved* = 0 and $h$.*moved* = 0) **then**
80: State := *FOLLOWER*
81: **else** State := *MSG*
82: **end if**
83: **end if**

84: **else if** State = *WAIT* **then**
85: **if** meet a *R_HEAD* agent **then**
86: State := *RENDEZVOUS*
87: Change state of *FOLLOWER* agents to *RENDEZVOUS*
88: **else** Wait (1)
89: **end if**
90: **end if**
91: *first_clock*:= 0

**Theorem 1.** *The CollectAgents algorithm solves the rendezvous problem for $k > 2$ agents in an unoriented ring of size $n$ with a special node $\otimes$, and with one malicious agent $M$. It returns a failure message when $n$ is odd, $k$ is even and there are at least two agents for each possible CW orientation.*

*Proof.* All the agents start moving at the same time in their *CW* direction, but there is no common sense of direction thus some agents could move in one direction while some others are moving in the opposite one. We have three possible cases: *1*) all the agents move in the same direction; *2*) all the agents except one move in the same direction; *3*) at least two agents move in one direction and at least two move in the opposite one.

*1*) We want to prove that in this case exactly one agent becomes *STOPPED*. Let us first assume by contradiction that no agent becomes *STOPPED*. There are two cases: a) Either $M$ stops or moves in a direction opposite to the other agents, but in this case it blocks one agent that becomes *STOPPED*, thus a contradiction; b) or $M$ keeps on moving in the same direction chosen by the agents, but in this case one agent will cross $\otimes$ twice and will stop, thus blocking $M$. Therefore, at least another agent will eventually be blocked by $M$ (no other agent stops at $\otimes$), and will become *STOPPED*, thus a contradiction also in this case. Let us now prove that the *STOPPED* agent is unique. Given that by hypothesis there are $k > 2$ agents that move in the same direction, and at most one agent becomes *STAR*, then one agent $C$ has to meet $A$, the *STOPPED* one, becomes *MSG*, while $A$ changes its state to *HEAD*. Note that, all the other agents in state *INITIAL* cannot become *STOPPED*, but become *FOLLOWER*, given that they meet $A$ before being blocked by *M*. The agents $A$ and $C$ move in opposite directions. $C$ reaches *M* in the opposite side because there are no other *STOPPED* or *HEAD* agents, given that $C$ is the first agent that moves in this direction. Thus, the *STOPPED* agent is $A$ and is unique. Let us now prove that $C$ will coordinate the gathering. When $C$ comes back in state *R_MSG* has the default *moved* value equal to $-1$. Agent $A$ in the opposite side continues to move in the same direction, with some agents in state *FOLLOWER*. However, the two agents $C$ and $A$ eventually meet, in fact either $A$ is stopped by $M$, or after it has reached $\otimes$ for the second time, it will remain stopped, together with the agents in state *FOLLOWER*, waiting for the arrival of $C$ to rendezvous. Note that, if one agent had previously stopped in $\otimes$, then it has to be in the path followed by *R_MSG*, thus it will become a *FOLLOWER* of *R_MSG* and will rendezvous with the other agents. Thus, if all agents initially move in the same direction rendezvous is achieved.

*2*) Let us assume that all the agents except one, called $B$, move in the same direction. The agent *M* cannot escape both from $B$ and from the agents going in the opposite direction, let us call this group $G_1$, thus *M* has to block both $B$ and another agent $A$ of $G_1$. $B$ becomes *STOPPED* but not *HEAD* because no other agent can reach it in state *INITIAL*, whereas $A$, becomes *STOPPED*, and once it is reached by an *INITIAL* agent $C$, $A$ becomes *HEAD*, and $C$ starts moving in the opposite direction, i.e. towards $B$, in state *MSG*. All the remaining agents of $G_1$ become *FOLLOWER* of $B$. When $C$ and $B$ meet, $B$ becomes a *FOLLOWER* of $C$, and $C$ moves in state *R_MSG* towards $A$ with the default *moved* value equal to $-1$. The agent $A$ and its *FOLLOWER* continue to move in *CW* direction, however $A$ is either blocked by $M$, or it stops after reaching $\otimes$ twice. When $C$ and $B$ reach $A$, all the other possible agents already reached $A$, and since $moved = -1$ they rendezvous.

*3*) At least two agents have an orientation in one direction, and other two in the other direction. Thus, exactly 2 agents become *HEAD* and 2 agents become *MSG*. Let us call them $A$ and $C$ the *HEAD* and the *MSG* in on one side, and $B$ and $D$ on the other side, respectively. All the other agents become *FOLLOWER* of the *HEAD* they meet. $C$ and $D$ perform a tour in opposite directions: they reverses direction, reach $B$ and $A$, respectively, and come back to $A$ and $B$ in state *R_MSG*, delivering the *moved* and the *parity* information related to the other *HEAD* agent.
We now have to show that the two groups of agents are able to meet. First observe that since agents are synchronous and links are FIFO $D$ ($C$) meets $A$ ($B$) in state *MSG* before that $C$ ($D$) comes back in state *R_MSG* delivering the information on the parity or oddness (computed using the *parity* bit) of the opposite group of agents, and the information on the possible movement of the opposite *HEAD*, stored in a variable *moved*. Note that, when *M* is surrounded, $A$ and $B$ are not able to move. We have two cases:

**- *k* is odd:** If $k$ is odd, the *HEAD* agent of a group with odd agents, i.e., with *parity* = 1, remains stopped in state *WAIT*, whereas the other *HEAD* agent, with *parity* = 0, starts moving in opposite direction in state *R_HEAD* with all the other agents of its group in state *FOLLOWER*. This group of agents will reach the other stopped group -regardless of the value $n$- and all agents will rendezvous.

**- *k* is even:** This case is more complicated and rendezvous can be achieved only when $M$ has been surrounded, i.e., when $A$ and $B$ cannot move anymore. This is possible since even if *M*, is very fast, due to the synchronicity of the agents, can block only $A$ or $B$ in one time step. Note also that each *HEAD* agent cannot meet $\otimes$ twice since *M* is surrounded before each of them visits all the nodes of the ring.
Thus, we have now to prove that $C$ and $D$ perform a same number of tours and eventually meet, together with their *FOLLOWER* agents. Given that $C$ and $D$ collect both the *moved* value of their *HEAD* and the one of the opposite *HEAD* agent, if at least one of the *moved* values is not 0 (which means at least one of the *HEAD* agents moved) they start a new round, otherwise they try to rendezvous. Observe that collecting both values and having FIFO links implies a synchronization of the rounds.
Let $G_1$ and $G_2$ be the two groups of agents surrounding *M* that will eventually be formed at the node of $A$ and of $B$ respectively. The *sync* variable changes at each clock (line 1), and is used to synchronize the movements. In the case $n$ even, $A$ and $B$ are separated by an odd number of nodes. If one *HEAD* agents changes direction at an even clock and the other at an odd clock (defined by the *sync* variable), only one of the groups waits one cycle of clock (line 26), and so the two groups move at an instant of time that preserves the odd distance and thus rendezvous. Without this synchronization agents would cross on an edge without meeting.
Conversely, if $n$ is odd $A$ and $B$ do not meet along the path are are blocked again by *M*, thus they move to the state *FAILURE* since they did not rendezvous. □

The complexity of the algorithm is as follows (the proof is in the Appendix).

**Theorem 2.** *The CollectAgents algorithm in an unoriented ring of size $n$ requires constant memory and it converges in $O(n)$ time steps and with $O(kn)$ total moves.*

**Strategy for $M$.** We now very briefly illustrate the best strategy for the malicious agent $M$, in order to delay as much as possible the gathering of the honest agents. Thus, we

will be considering time constraints. If $k-1$ agents have the same *CW* orientation, *M* moves *CW* up to when it is blocked by an agent and stops. If $k$ agents have the same orientation, *M* moves in that direction up to when it is blocked by an agent, then it waits up to when agent moves (as it has become a *FOLLOWER* of an *R_MSG*), and it follows it, until it is blocked again and stops. This happens after a *HEAD* has crossed twice the node $\otimes$ and has become *STOPPED*. Finally, if agents are split in two groups, and $k$ is odd, it moves in the direction of the group of even size, if $k$ is even it alternatively blocks *HEAD* agents from one or the other side, so that both *HEAD* agents start a round as soon as possible with the effect of maximizing the number of rounds. See Algorithm 2 in the Appendix for more details and also the proof of the following theorem.

**Lemma 2.** *Algorithm 2 provides the best strategy for the malicious agent to delay as much as possible the gathering of the honest agents.*

**Results for the oriented rings, and for the case $k = 2$ in unoriented rings.** For lack of space, we just very shortly illustrate two new cases, more details may be found in [12]. Let us first consider the case of the oriented ring. First note that there are no unsolvable cases, given that the symmetric configurations of Lemma 1 never arise when agents move in the same direction. One solution for $k \geq 2$ would be to apply Algorithm 1 of [5] which might require that the agents reverse direction three times when they are blocked by *M*. Moreover agents will terminate but do not know when there is global termination. Another solution for $k \geq 3$ would be to apply Algorithm *CollectAgents* in the case in which all the agents move in the same direction. This solution could even be improved: After an agent has become *HEAD* it just stops and waits, since if all the agents move in the same direction, then *HEAD* will not need to surround $M$. Finally, we have devised another simpler algorithm that requires less rounds, assures global termination, works for $k \geq 2$ agents, and uses only constant memory. The general idea is as follows: Suppose all the agents start moving in the *CW* direction, then the first agent that is blocked by *M*, let us call it $A$, becomes the leader and starts moving in the *CCW* direction. Then, when $A$ meets an agent that is moving in the *CW* direction, let us call it $B$, $B$ stops, while $A$ continues to move until it is blocked again by *M*. Then, $A$ reverses direction and moves back. The rendezvous is achieved at the node of $B$. Note that, also in this case, an agent that arrives in $\otimes$ stops to block *M*. Finally, to assure that when $A$ moves in the *CCW* direction it meets any other agent, e.g. $B$, that is moving in the *CW* direction, we assume that $A$ waits at the even clocks and moves at the odd clocks, while $B$ moves at the even clocks and wait at the odd clocks.

Let us now consider the case $k = 2$ in the unoriented ring. Note that, in the unoriented ring with asynchronous agents the problem is unsolvable for any even number $k \geq 2$, of honest agents, even if the agents know $k$ [5]. With synchronous agents we have the following. For $n$ odd Lemma 1 proofs that the problem is not solvable, for $n$ even we now briefly discuss a solution that requires $O(\log n)$ bits of memory, and assumes that the agents know that $k = 2$. The algorithm works as follows. It uses the general idea the Controlled Distance algorithm in a ring [14]. The agents move back and forth for a certain distance $x$ which increases at each round, in particular we choose $x = 2^i$, for $i = 0, 1, 2, \ldots$. At each round $i$ both agents, that we call $A$ and $B$, try to move for $x$ steps in the *CW* direction, i.e., for $x$ cycles of clock either they move, or

they wait if they are blocked by $M$. Then, they move back in the *CCW* direction for $x$ steps, or they wait if they are blocked by *M* in the opposite direction. During this walk if they meet each other they gather. Note that, if they have waited some time moving *CW*, then they might go back passing the starting position. It is possible to prove that eventually the two agents will either meet at some intermediate node, or they will eventually surround *M* and, since they are synchronized, and they start moving in *CCW* direction at the same time step, they will meet at the middle node, opposite to *M* in the ring. To prevent agents to infinitely increase value $x$ when they start in the same direction, an agent stops when it reaches $\otimes$ twice so that the other agent will meet it at the node $\otimes$.

## 4 Implementation

In this section we illustrate a proof-of-concept implementation based on the Lego Mindstorms EV3 platform,[4] which is widely adopted in robotic courses.

We have represented each undirected edge of the ring as two parallel links, so that robots moving in two opposite directions on the same edge do not collide. The nodes are all identical, and they have been represented as circles that can be traversed by the agents moving around the border. The special node is labeled with a yellow marker. A picture of the graph used during our experiments is provided in Figure 1a in the Appendix. The honest robots are all identical, whereas the malicious agent has been physically modified to be easily noticeable, as shown in Figure 1b in the Appendix.

To built our robots we have used standard Lego EV3 components. We have used the *EV3 Color Sensor* to detect the lines representing the edges of the graph so to let the robots move along them. We have used the *EV3 Infrared Seeking Sensor* and *EV3 Ultrasonic Sensor* to detect the other honest agents in the same node, and to avoid collisions when robots are following each other. By rotating the sensors on top of the *EV3 Servo Motors* we are also able to detect robots placed crosswise.

Face to face communication is implemented using the built-in Bluetooth adapters. However, given the small physical dimension of the ring we could not use the Received Signal Strength Indicator (RSSI) of the Bluetooth devices to connect to the robots in the same node, since all devices were showing very similar signal strengths. We have then used a centralized server that mediates connections and only enables face to face communication in the same node. We claim that on a bigger network the strength indicator would work more reliably and local communication might be implemented without resorting to a centralized server.

The software platform used for the implementation is *ev3dev*,[5] an open-source Linux-based operating system fully compatible with Lego Mindstorm robots. EV3 hardware can be controlled from a wide range of common programming languages providing bindings for the ev3dev device API. We decided to implement our algorithms in Python3 since it offers high code readability and enables rapid prototyping of applications. The source code written to evaluate the protocols consists of around 600 lines and

[4] Lego mindstorms ev3, 2016. http://www.lego.com/en-us/mindstorms/products/31313-mindstorms-ev3

[5] Lego EV3 compatible operating system, http://www.ev3dev.org/

is publicly available at our github repository page,[6] together with some exemplifying videos.[7]

## 5 Conclusions

In this paper we have presented the problem of gathering a set of mobile agents in presence of mobile transient faults, represented by a mobile malicious agent. We have studied the problem in oriented and unoriented ring networks, and we have show that the proposed solutions are realistic by giving a proof-of-concept implementation of the algorithms with real Lego Mindstorms EV3 robots. We are presently studying the problem of gathering in an unoriented ring with $k = 2$ agents and constant memory, and the gathering of synchronous agents in other topologies.

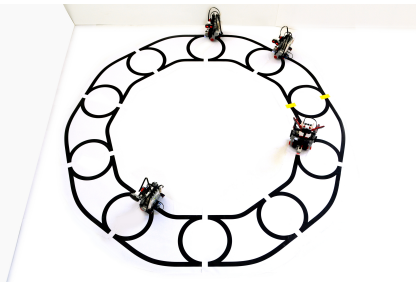
## References


1. S. Alpern and S. Gal. Searching for an agent who may or may not want to be found. *Operations Research*, 50(2):311–323, 2002.
2. J. Chalopin, Y. Dieudonne, A. Labourel, and A. Pelc. Rendezvous in networks in spite of delay faults. *Distributed Computing*, 29:187–205, 2016.
3. R. Cohen and D. Peleg. Convergence of autonomous mobile robots with inaccurate sensors and movements. *SIAM J. on Computing*, 38:276–302, 2008.
4. J. Czyzowicz, A. Labourel, and A. Pelc. How to meet asynchronously (almost) everywhere. In *Proc. of 21st Annual ACM-SIAM Symp. on Discrete Algorithms*, 2010.
5. S. Das, F.L. Luccio, and E. Markou. Mobile agents rendezvous in spite of a malicious agent. In *11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (Algosensors 2015)*, LNCS 9536, pages 211–224. Springer, Patras, Greece 2015.
6. S. Das, M. Mihalak, R. Sramek, E. Vicari, and P. Widmayer. Rendezvous of mobile agents when tokens fail anytime. In *OPODIS*, LNCS 5401, pages 463–480, 2008.
7. Y. Dieudonne, A. Pelc, and D. Peleg. Gathering despite mischief. *ACM Transactions on Algorithms*, 11(1):1, 2014.
8. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48(1):67–90, 2007.
9. P. Flocchini and N. Santoro. Distributed security algorithms for mobile agents. In Jiannong Cao and Sajal K. Das, editors, *Mobile Agents in Networking and Distributed Computing*, chapter 3, pages 41–70. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2012.
10. R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary graphs. *TCS*, 384(2-3):201–221, 2007.
11. F. L. Luccio. Contiguous search problem in sierpinski graphs. *Theory of Comp. Sys.*, (44):186–204, 2009.
12. Davide Moro. From theory to practice: Solving the rendezvous problem using real robots. Master's thesis, DAIS, Università Ca' Foscari, Venezia, June 2016.
13. A. Pásztor. Gathering simulation of real robot swarm. *Tehnicki vjesnik*, 21(5):1073–1080, 2014.
14. Nicola Santoro. *Design and analysis of distributed algorithms*, volume 56. John Wiley & Sons, 2006.

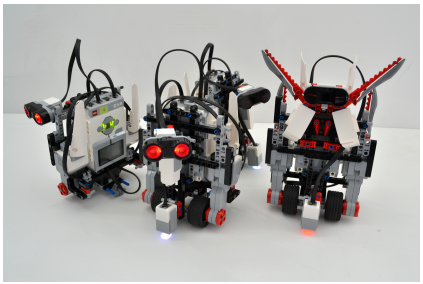

[6] Source code https://github.com/secgroup/MTFGatheRing

[7] Video of the rendezvous in an unoriented ring, https://github.com/secgroup/MTFGatheRing/raw/master/videos/robots_unoriented.mp4

## Appendix



(a) Representation of the graph.



(b) Two honest and one malicious robot.

Fig. 1: The Lego Mindstorms EV3 setup.

**Proof of Theorem 2**

*Proof.* The amount of memory used by the agents is constant since it does not depend on $n$ or $k$, but it trivially depends on a constant number of variables which can only take constant values.

The worst case time complexity has to be computed on the three possible settings described in the proof of Theorem 1. In the first two cases, i.e., when all the agents are moving in the same direction, or when only one moves in the opposite direction, we have that after at most $n-1$ steps each agent arrives either at *STAR* or is blocked by $M$. In the first case $M$ might keep on moving around but after at most other $n$ steps at least on agent stops in *STAR*. In all the other cases one agent stops is a shorter time, blocked by $M$. Then, one agent moves back and forth to collect agents in at most $2\times(n-1)$ time steps, and then rendezvous. Globally, in these two cases $O(n)$ time steps are required in the worst case.

The third case, the one in which at least two pair of agents move in opposite directions, is more complicated. The first round has the same complexity of the second case, i.e., $O(n)$ time steps. Then, some rounds are repeated, each of which assumes that a message goes back and forth from one *HEAD* to the opposite one. The worst case is the one in which $M$ block the agents as much as possible before it is surrounded. Assume the extreme case in which all agents start in neighbouring positions and recall that at each round $M$ might only block the agent on one side. Thus, when the distance among the agent is $x$, while the *MSG* are moving, the two heads conquer $x$ positions, thus a bound is on the number of times that a *MSG* agent turns, is given by $\sum_{i=0}^{\log(n)} 2^i = 2n-1$. Then some few extra rounds are required to collect the final information or to fail. Globally the time is $O(n)$ time steps.

A trivial bound on the number of moves is given by $O(k \times n)$ if each agent has execute all steps (i.e., when agents are in contiguous positions and to move together towards $M$, starting from the furthest positions. □

**Algorithm 2** Blocking strategy for *M* in the unoriented ring with $k > 2$

```
# Algorithm for the Malicious Agent in Unoriented Ring
# Assumptions: One node is marked ⊗, all honest agents simultaneously wake up

if k agents have initially the same CW orientation then
    while not blocked do
        move in CW direction
    end while
    while blocked do
        Wait (1)
    end while
    while not blocked do
        move in CCW direction
    end while
    State:=STOPPED
else if k − 1 agents have initially the same CW orientation then
    while not blocked do
        move in CW direction
    end while
    State:=STOPPED
else if k is odd then
    while not blocked do
        move in the direction of the group of even size
    end while
else if k is even then
    if not Blocked in both directions then
        Blocks in one move a HEAD agent in one direction and, in the next move, the other
    end if
end if
```

**Proof of Lemma 2**

*Proof.* The correctness of the algorithm derives from the fact that if all the agents move in the same *CW* direction, for two rounds do not take any action unless they are blocked by $M$, so $M$ tries to delay the blocking as much as possible. Once *M* is blocked it waits and it tries to delay the blocking of the leader moving in *CCW* as much as possible. If only one agent moves in the opposite direction the best strategy for *M* is to try to escape from the $k-1$ agents since it will be one of this group to perform the gathering after moving *CCW* and *CW*. Thus, this step has to be delayed as much as possible. If the agents move in opposite directions, and $k$ is odd, $M$ tries to delay as much as possible the agent that collects the group of even size. The global time will be given by the time the second *INITIAL* agent meets a *STOPPED* agent that has been blocked by $M$ (so this has to be maximized), the time the *INITIAL* agent that now becomes *MSG* spends to reach the group of even size ($n-1$ time steps), and other $n-1$ steps with this group to go back. The last case is $k$ even. In this case to maximize the gathering time $M$ has to delay the meeting phase in the middle position, thus at each time step it blocks one of the *HEAD* agents. □