

Gathering of Robots in a Ring with Mobile Faults

Shantanu Das¹, Riccardo Focardi², Flaminia L. Luccio²,
Euripides Markou³, Davide Moro², and Marco Squarcina²

¹ Aix Marseille Univ, CNRS, LIF, 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 un-oriented 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

Copyright © by the paper's authors. Copying permitted for private and academic purposes.

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 122–135 published in CEUR Workshop Proceedings Vol-1720 at <http://ceur-ws.org/Vol-1720>

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]. The model used in [2] is somehow similar to ours in the sense that the adversary may delay for an arbitrary, but finite time, the

movements of the agents. However, contrary to the model of [2], in our setting, one of 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 usefulness 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, \dots, 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 either oriented or unoriented ring networks $R = (V, E)$, with $|V| = n$ nodes. The links incident to a host are distinctly labeled. In the oriented ring, the links are labelled consistently to allow all agents agree on a common sense of direction. In the unoriented ring there is no such agreement and the labels on the links are arbitrary. 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 , and can place itself in \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. Moreover, recall that $n - 1$, the number of free positions is even, i.e., there is no meeting point in the middle opposite to \otimes , and therefore the agents of the two groups 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, those that move in one direction and those that move in the opposite direction. Note that the agents are not aware of this information so they cannot adapt their behaviour accordingly.

If an agent continues moving in the same direction and eventually reaches \otimes for the second time, then the agent can be sure that all agents are moving in the same direction. In this case, one of the agents - the first agent to reach \otimes twice - will stop at \otimes . The other agents continue moving in the same direction until they are blocked by M . The first agent to be blocked stops (in state *STOPPED*) and all other agents gather at this node, except one of the agents (the first one to meet the *STOPPED* agent) which changes to state *MSG* ("messenger") and goes around the ring in the other direction until it is blocked by M . At this point M is surrounded by the stopped agent (now in state *HEAD*) and the messenger agent. The messenger agent (changing to state *R_MSG*) returns back, collecting the *STAR* agent on the way, and rendezvous is achieved when these two agents reach the other agents who are waiting at the *HEAD* agent.

When the agents do not have the same orientation then the situation is different. In this case there are two groups of agents, G_1 and G_2 , moving in

opposite directions. One agent from each group will eventually be blocked by M and become the “leader” (*HEAD*) agent for their group. The two groups of agents will start gathering in two distinct locations. To bring the two groups together, we use the fact that either k is odd or n is even. In the former case, one of the groups is of odd size, and agents of this group turn back and go to the other group. Otherwise if k is even and both groups are of even size, then the leader (*HEAD*) agents of the two groups try to push M from two sides until M is cornered in one node of the network. The segment of the ring containing the other nodes is of odd size and the two groups of agents can meet at the center of this segment. Let agent A for G_1 , and agent B for G_2 , be the agents that are blocked by M and become *STOPPED* (line 6). When two other agents, say C and D , meet A and B respectively 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. When agents C and D 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. If either G_1 or G_2 is of odd size and all the agents of the odd group move towards even group (line 23), and rendezvous is achieved. If k is even, and both G_1 and G_2 are of even size, then agents C and D go back and forth, until 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 or the algorithm fails if n is odd (line 38).

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:  $sync := (sync + 1) \bmod 2$ 
2: if State = INITIAL then
3:   dir := CW
4:   if  $first\_clock = 1$  and Reached  $\otimes$  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  $\otimes$  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  $parity := (parity + 1) \bmod 2$ 
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  $\otimes$  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. *Algorithm CollectAgents 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 of the two possible orientations.*

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 A . 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 $FOLLOWER$ agents 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 $moved = -1$ the agents terminate in state $RENDEZVOUS$.

(3) At least two agents have an orientation in one direction, and another 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 on one side, and B and D , those 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 reverse 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 $parity = 1$, 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, M can block only A or B in one time step. Note also that none of the $HEAD$ agents can meet \otimes twice since M is surrounded before either of them could visit all the nodes of the ring.

Thus, we have now to prove that C and D perform a same number of turns and they 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 case n is even, A and B are separated by an odd number of nodes. If one $HEAD$ agent changes direction at an even clock tick and the other at an odd clock tick (defined by the $sync$ variable), only one of the groups waits one clock cycle (line 26), and so the two groups move at an instant of time that preserves the odd distance and thus,

they rendezvous. Without this synchronization, the agents may cross on an edge without meeting.

Conversely, if n is odd A and B do not meet along the path and are blocked again by M , thus they move to the state *FAILURE* since they did not rendezvous. \square

The complexity of the algorithm is as follows.

Theorem 2. *Algorithm *CollectAgents* 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 all k agents have the same orientation, M moves in that direction up to when it is blocked by an agent, then it waits until the agent moves (as it has become a *FOLLOWER* of an *R_MSG*), and M follows the agent 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, M moves in the direction of the group of even size; if k is even M 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. Let us call this algorithm *BlockAgents* for M .

Lemma 2. *Algorithm *BlockAgents* 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 . 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 clock ticks and moves at the odd clock ticks, while B moves at the even clock ticks and wait at the odd clock ticks.

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 proves 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 of 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 $i = 0, 1, 2, \dots$. 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. The honest robots are all identical, whereas the malicious agent has been physically modified to be easily noticeable.

To build 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

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

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 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 shown 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 Lego EV3 compatible operating system, <http://www.ev3dev.org/>

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

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.