Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Gathering of robots in a ring with mobile faults ☆

Shantanu Das [a], Riccardo Focardi [b], Flaminia L. Luccio [b,*], Euripides Markou [c], Marco Squarcina [b]

[a] LIF, Aix-Marseille University, avenue de Luminy 163, F-13288 Marseille Cedex 9, France
[b] DAIS, Università Ca' Foscari Venezia, via Torino 155, 30170, Venezia, Italy
[c] DCSBI, University of Thessaly, Papasiopoulou 2, 35131 Galaneika, Lamia, Greece

ARTICLE INFO

Article history:
Received 8 May 2017
Received in revised form 5 March 2018
Accepted 1 May 2018
Available online xxxx

Keywords:
Ring network
Mobile agents
Gathering problem
Malicious agent
Robots

ABSTRACT

This paper studies the well-known problem of *gathering* multiple mobile agents moving in a graph, but unlike previous results, we consider the problem in the presence of an adversarial mobile entity which we call the *malicious agent*. The malicious entity can occupy any empty node and prevent honest mobile agents from entering this node. This new adversarial model is interesting as it models transient mobile faults that can appear anywhere in a network. Moreover, our model lies between the less powerful *delay-fault model*, where the adversary can block an agent for only a finite time, and the more powerful but static fault model of *black holes* that can even destroy the agents.
We study the problem for ring networks and we provide a complete characterization of the solvability of gathering, depending on the size n of the ring and the number of agents k. We consider both oriented or unoriented rings with either synchronous or asynchronous agents. We prove that in an unoriented ring network with asynchronous agents the problem is not solvable when k is even, while for synchronous agents the problem is unsolvable when both n is odd and k is even. We then present algorithms that solve gathering for all the remaining cases, thus completely solving the problem. Finally, we provide a proof-of-concept implementation of the synchronous algorithms using programmable Lego Mindstorms EV3 robots.

1. Introduction

1.1. Overview

We consider the problem of gathering of multiple mobile agents or rendezvous of two agents moving in a network which is modeled as an undirected graph. The mobile agents are autonomous entities that can move along the edges of the graph, each executing its own algorithm. The agents could represent either physical mobile robots moving in a terrain or software agents moving in a network of computers. The objective is to gather all agents at the same location (i.e., at a single node

☆ Preliminary versions of this work appeared in [14,13].

* Corresponding author.
E-mail addresses: shantanu.das@lif.univ-mrs.fr (S. Das), focardi@unive.it (R. Focardi), luccio@unive.it (F.L. Luccio), emarkou@ucg.gr (E. Markou), squarcina@unive.it (M. Squarcina).
URLs: http://pageperso.lif.univ-mrs.fr/~shantanu.das/en/ (S. Das), http://dsi.unive.it/~focardi (R. Focardi), http://dsi.unive.it/~luccio (F.L. Luccio), http://emarkou.users.uth.gr/ (E. Markou), https://minimalblue.com/ (M. Squarcina).

https://doi.org/10.1016/j.tcs.2018.05.002
0304-3975/

of the graph). Gathering is a crucial task for teams of autonomous mobile robots, since they may need to meet in order to share information or coordinate. Therefore this problem has been widely studied for different classes of graphs such as ring, torus, hypercube or arbitrary graphs, under different assumptions about the mobile agents, synchronous or asynchronous, with or without identities and so on. In this paper, we consider agents that are identical (i.e., anonymous) and execute the same algorithm. Furthermore, we are interested in solutions that are scalable to graphs of any size; thus we restrict the size of the memory of the agent to a constant independent of the size of the graph. The mobile agents in this paper are finite automata moving in a graph, with only local visibility; an agent can only see its current node and the edges incident to it.

Gathering of agents in a graph is a well studied problem with wealth of literature providing different techniques under various models and optimization criteria (see e.g. the book [2] or the survey [36]). The majority of these results are for fault-free environments. In recent years, some researchers have considered the gathering problem in the presence of faults. In a network, possible faults can be permanent failures of nodes or edges. Gathering has been studied in the presence of a dangerous node (representing a crashed node), called a *black hole* that destroys any agents arriving at that node. However in this case not all agents can gather and the objective is to gather the maximum number of agents possible by first locating the black hole node (see, e.g., [20,30]) and then avoiding it while gathering [5,6]. When the fault is not localized and can appear at different locations in the graph, such fault is modeled as a mobile hostile entity (an *intruder*) that moves arbitrarily fast through the graph. Previous work on this fault model involves designing strategies for capturing the intruder agent using a team of honest agents. An example of this kind is the so-called *network decontamination* or *intruder capture* problem (see, e.g., [26,33]). Note that the hostile agent in this case can not harm the honest agent and the objective of the honest agents is to meet the hostile agent, while the hostile agent tries to escape from the honest agents. This can be seen as an example of asymmetric gathering. Gathering has been studied also in the presence of *Byzantine agents*, which may behave in an arbitrary manner and may also provide false information to the honest agents so to prevent their gathering [18]. In this case, the objective is to gather only the honest agents and the main difficulty is to distinguish the Byzantine agents from the honest agents. Another type of failures that has been studied in the context of gathering is the co-called *delay faults* in networks [8], where an adversary can delay the movement of a honest agent for an arbitrary but finite time.

In this paper, we introduce a different type of mobile fault which we call the *malicious agent*. This agent can physically block the movement of a honest agent to the node it occupies, preventing the honest agents from gathering in some cases. As in the case of intruder agents, the malicious agent 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 may even have full knowledge of the actions that will be taken by the honest agents. On the other hand the malicious agent can be detected by a honest agent, when it blocks the path of this honest agent. Note that the malicious agent in our paper is a stronger adversary than the Byzantine agent or the Intruder agent, or the delay faults. Indeed we can show that even one malicious agent can prevent gathering of honest agents in many cases. In particular if the malicious agents can permanently occupy a cut set of nodes in the graph, they can retain the honest agents in two separate parts of the graph, thus preventing gathering [14]. For a single malicious agent, the graph must be at least bi-connected to prevent this scenario. In this paper, we restrict our study to ring networks with exactly one malicious agent. The ring is one of the simplest biconnected topologies which can be traversed even by constant-sized memory agents. On the other hand, the high symmetry of the network makes it very challenging to solve gathering in the ring [31]. We investigate the feasibility of gathering for $k \geq 2$ honest agents in oriented and unoriented ring networks in the presence of the malicious adversary, that blocks other agents from having access to parts of the network. Our objective is to study the feasibility of gathering with minimal assumptions. The honest agents are identical (thus anonymous), and they only have local communication capabilities allowing two agents to talk only when they are in the same node. Furthermore since the honest agents have only constant-sized memory, they do not know the size of the ring and the number of agents present in the ring.

Gathering of identical agents in a ring is impossible due to symmetry, even if the agents have infinite memory and know the size of the ring and the number of agents. This holds even for the scenario without any faults. Therefore, all solutions to gathering use symmetry breaking mechanisms such as assuming unique identities or distinct speeds, or allowing marking of nodes with tokens etc. In this paper, we break symmetry by assuming a specially marked node in the ring which can be used as *landmark*. The use of a landmark node is the simplest mechanism for symmetry breaking that guarantees an immediate solution in fault free networks. However, in the presence of a malicious agent, the gathering problem still remains challenging as the malicious agent may prevent the honest agents from ever reaching the landmark node, if the malicious agent occupies this node. Therefore, any trivial solution to the problem asking all agents to go to the landmark, would not work. In this paper, the use of the landmark node allows us to focus on the problem of tolerating the malicious agent (the main issue in this work), rather than on the issue of symmetry breaking using identities (which has been extensively studied before).

1.2. Contributions

We provide a characterization of the feasible instances for gathering of k agents in a ring of size n, in presence of a malicious agent. (1) In an oriented ring, we show that gathering is always possible and we provide a gathering algorithm for any $k \geq 2$. (2) In an unoriented ring network of n nodes we show that gathering of asynchronous agents is not solvable if k is even while, gathering of synchronous agents is unsolvable when both n is odd and k is even. Surprisingly we show that all the other cases in an unoriented ring are solvable for $k > 2$ agents and provide algorithms for all those cases. In

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

particular, we give (i) an algorithm for asynchronous agents that works for any n whenever k is odd, and (ii) an algorithm for synchronous agents which works for all solvable cases, i.e., when either k is odd or n is even (assuming $k > 2$). For the special case of $k = 2$ agents in an unoriented ring, we prove that rendezvous is impossible for constant-sized memory agents; however we present a protocol for rendezvous of 2 synchronous agents when they have enough memory to count up to n.

With the exception of the algorithm for oriented rings (Algorithm 1), all other algorithms are 'global-termination' protocols in the sense that each agent is aware when all agents have gathered and terminated. The algorithm for oriented rings (Algorithm 1) is a 'local-termination' protocol where each agent terminates its execution locally and although gathering of all agents is achieved within a finite number of steps, an agent may not be aware whether all agents have gathered.

Finally, we show that the proposed algorithms and underlying assumptions are realistic through a proof-of-concept implementation using programmable *Lego Mindstorms EV3* robots.

1.3. Related work

The problem of gathering multiple autonomous agents has been extensively studied under two very different models. The first model considers agents with local visibility that move from node to node in a network (represented by a finite connected graph) and can see or communicate with other agents that are colocated at the same node. The objective is to gather all agents at a single node of the graph. The second model (called the *look-compute-move* model) considers mobile robots, represented as point objects, moving on a plane (or other geometric regions, e.g., a line or polygon), where each robot can see all other robots at any time (i.e., they have global visibility) and the objective is to gather the robots at a common point in the plane. This paper studies gathering in the first model and we consider here only results in this model. Results on gathering in the look-compute-model uses completely different techniques, based on the global snapshot of the environment (this includes some results on a hybrid model [17] where the environment is a graph but the robots still operate based on global snapshots as in the look-compute-move model); for such results the reader is referred to the book [25] and the references therein. Randomized algorithms for gathering two or more agents has been considered in [2] under various environments; such results are out of scope of this paper as we restrict ourselves to deterministic strategies only.

Deterministic gathering in the network model has been studied both in synchronous and asynchronous case for different graph topologies, e.g., see [36,7]. When there are no faults or failures, deterministic gathering can be solved relatively easily, even in asynchronous networks, when the network has some asymmetry (e.g., a distinguished node), since the mobile agents can simply be instructed to meet at such a distinguished node. However, this is not the case for symmetric networks and the gathering problem in such settings is non-trivial and not always solvable even in simple topologies such as the ring network; the book [31] studies different gathering techniques in ring networks. In general, symmetry-breaking for the gathering problem can be achieved by attaching unique identifiers to the agents (see, e.g., [12,39]), or in the anonymous case using tokens as in, e.g., [24]. Most of the above results consider that the agents have enough memory to remember the complete network once they have traversed it. Gathering of agents with small memory has been studied in [11] where the agents have memory logarithmic in the size of the network, which is known to be optimal for gathering [27] or even for exploring arbitrary unknown networks. Note that, agents with constant memory (as in this paper) can only explore specific families of graphs such as rings, tori, grids, trees, cliques etc. Algorithms for such agents has been studied in [5,32,10].

With respect to hostile or faulty environments, the gathering problem has been studied in the presence of a so-called *black hole* which is a stationary fault that destroys agents entering a particular node (or link) of the network [6,19]. Less severe stationary faults which can be repaired with the sacrifice of at least one agent per faulty node, have been studied in [9,16], with the objective of detecting and repairing all faults. When the fault is not stationary but mobile, we can represent the environment by a dynamic network where the set of available links can change with time. Gathering in such networks has been studied, e.g., in [34,38] for the case of rings. A well studied problem in the context of a mobile adversary is the problem of graph searching where a team of mobile agents has to decontaminate the infected sites and prevent any re-infection of cleaned areas. This problem is equivalent to the one of capturing a fast and invisible fugitive moving in the network, which is often called *asymmetric rendezvous* as the two parties have opposing objectives unlike the gathering problem. For results on this and related problems see, e.g., [3,21,22,33]. Another fault model that has been considered is the failure of the marking devices (tokens) used by the agents [15,23,29].

Some recent results have considered failures of the agents themselves. As mentioned, the recent paper [18] considers the problem of gathering in the presence of *Byzantine agents*, which are indistinguishable from the *honest* agents, but may behave in an arbitrary manner thus trying to prevent the gathering of honest agents. Another related work includes gathering in networks with delay faults [8]. The model used in [8] is somehow similar to ours in the sense that the adversary may prevent a honest agent from moving for an arbitrary, but finite time. However, contrary to our model, in the model of [8], no agent may be prevented forever from moving.

For the gathering of robots in geometric settings under the *look-compute-move* model, failures of robots have been considered in some recent work. In particular, gathering of mobile agents on the plane has been studied when there are faulty agents which may crash [1,4]. From the view of practical implementations of gathering algorithms, [35] proposes an example of a simulation of a known distributed fault-free gathering algorithm, using real NXT robots. Another practical issue is considered in [28] where the authors propose a new model for sensing devices based on embedded omnidirectional cam-

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

eras, sonars, or lasers that are used on real robots moving on a plane. Under this new model where the visibility of robots may be asymmetric, the authors study the algorithmic challenge of gathering four voluminous robots in the asynchronous setting.

1.4. Paper structure

The paper is organized as follows: In Section 2 we illustrate the formal model for honest and malicious agents. In Section 3 we present an algorithm for $k \geq 2$ synchronous and asynchronous agents in oriented rings. We then consider unoriented rings of size $n > k$, and in Section 4 we present some impossibility results, while in Section 5 we show two different algorithms for $k > 2$ agents, one that solves the problem for asynchronous agents when k is odd, and the other that solves gathering of synchronous agents if k is odd or n is even. These algorithms together solve all feasible instances of gathering except for the case of two synchronous agents in unoriented rings. In Section 6 we study the rendezvous problem with $k = 2$ synchronous agents in unoriented rings. We first present an impossibility result for agents with constant-sized memory, and we then provide a solution for agents with $O(\log n)$ memory. In Section 7 we present the implementation of the algorithms on programmable robots, and we discuss the usefulness of the experimentation with the real robots in the development of the theoretical models. We conclude in Section 8 with some discussion for future research.

2. The model

We consider a distributed network modeled by an undirected, connected graph $G = (V, E)$ where V is a set of anonymous and identical nodes or *hosts*, and E is a set of edges or connections between nodes. We assume that in the network there are k *honest* anonymous identical synchronous/asynchronous 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 assumptions, 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 labeled in a globally consistent way to allow all agents to agree on a common sense of direction. In the unoriented ring the labels on the links are only locally consistent in the sense that two agents located at different nodes might move in different directions if they exit their nodes by taking the same port-label and therefore in this case, not necessarily all agents perceive the same direction as clockwise. The edges of the network are FIFO links, meaning that all agents, including the malicious one, that move on the same link arrive at the other end of the link, respecting a first-in-first-out ordering. Thus, one agent will pass another agent only if the latter deliberately stops in a node. We call a node v *occupied* when one or more *honest* agents are in v, and we call v *free* or *unoccupied* otherwise. Exactly one node of the ring is marked $\otimes$, and all the other nodes are identical and indistinguishable from each-other.

Honest agents The agents are independent, identical and anonymous processes with some internal memory of constant size. Hence an agent can be seen as a Finite State Automaton with a constant number of states and a constant-sized space in which can store information. These agents are initially scattered in the graph, i.e., at most one agent at a node, start the execution of the protocol at the same time, and can move along the edges of G. An agent located at a node v can communicate (i.e., exchange states and (constant-sized) memory information) with any other agent that is located at v. An agent cannot see or communicate with any agent that is not located at the same node, i.e., communication is face to face and visibility is restricted to the node occupied by the agent. 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 opposite directions do not notice each other, and cannot meet on the edge. Their goal is to gather 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 properties of **synchronous** agents are 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; (d) in each time step each agent performs the following actions: An agent arriving at a node v through a port q performs some computation based on its own state, the states of other agents at the node, the incoming port number and the state of the node (occupied, unoccupied, $\otimes$). 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. Based on the computation, the agent changes its state and performs the operation Move(p), if $p > 0$. If p is the port number for the edge (v, z), and the node z is not occupied by the malicious agent then the agent arrives at node z in the next time unit. Otherwise, the agent remains at node v in the next time unit, with a flag set in its memory notifying the agent that the move was unsuccessful. In this case, we say that the agent was *blocked* by M. Each edge traversal (whether successful or not) takes one time unit for synchronous agents.

For **asynchronous** agents, we have the following differences compared to synchronous agents: An edge traversal (or attempted edge traversal) takes a finite but not a priori known time (selected by an adversary). The adversary may also delay the starting of an agent's move. We also emphasize that while an asynchronous agent is traversing an edge, it might not be found in any of its endpoints for some arbitrary time selected by the adversary. All other properties are the same as for synchronous agents.

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

Algorithm 1 *RV-OR*: Gathering of $k \geq 2$ agents in oriented rings.

```
# Assumptions: i := 0; DIR := CW;
# Start in state: Initial

repeat
    repeat
        Move(DIR);
    until ((blocked by M) or (arrived at ⊗) or (met a Stopped agent));
    i := i + 1;
    if you met a Stopped agent then
        Become Stopped agent;
    else
        if i = 1 or i = 2 then
            if Current node is ⊗ then
                Become Stopped;
            else if blocked by M then
                Reverse direction (DIR := inverse(DIR));
            end if
        end if
        if i = 3 then
            if Current node is ⊗ or last move was blocked by M then
                Reverse direction (DIR := inverse(DIR));
            end if
        end if
        if i = 4 then
            if Current node is ⊗ or last move was blocked by M then
                Become Stopped;
            end if
        end if
    end if
until you are a Stopped agent;
```

Malicious agent We consider a worst case scenario in which the malicious agent M is more powerful 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.

Notice that, since the ring is a highly symmetric anonymous network and the agents are also anonymous, gathering is impossible even if the agents have unlimited memory and know k, n, since an adversary can keep synchronized the agents so that they always take the same actions at the same time and therefore they maintain their initial distances (the malicious agent can keep on moving synchronized with the honest agents). Thus, in order to solve the problem we need to add some additional constraint to the model. A natural option is to assume that there is a special node labeled $\otimes$ in the ring which can be recognized by the agents. Note that, the malicious agent is so powerful that it could place itself on $\otimes$ and never move from there. Our strategies also work under this scenario.

3. Gathering in an oriented ring

In this section we propose an algorithm that solves the gathering problem in any oriented ring network with asynchronous or synchronous agents. We assume that the two incident ports at each node of the ring are labeled as clockwise (*CW* for short) or counter-clockwise (*CCW* for short) in a consistent manner so that all agents agree on the ring orientation.

The idea of the algorithm is the following: Each agent moves in the *CW* direction until it meets $\otimes$ or is blocked by M. For the first three times that the agent is blocked by M without meeting $\otimes$, it reverses its direction and continues moving in the opposite direction. Due to the FIFO property and the fact that the agent cannot pass over M, we can show that if an agent was blocked at least three times by M, then the other agents should have been blocked by M at least twice, without meeting the special node $\otimes$ (see Lemma 1). After an agent has been blocked by M three times, the next time it is blocked by M or meets $\otimes$ it stops. On the other hand, if the agent meets $\otimes$ before being blocked by M twice, then the agent stops at $\otimes$, and all the other agents will arrive at $\otimes$ after being blocked by M at most once. The algorithm called *RV-OR* is presented in the above table as Algorithm 1.

Lemma 1. *During the execution of Algorithm 1, if an agent is blocked by M in the fourth iteration of the* while *loop, then any other agent must have been blocked by M at least two times.*

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

Proof. Consider the first agent A that is blocked by M two times at nodes u, v without meeting $\otimes$ and consider the segment of the ring between these two nodes u and v, where A was moving until it is blocked by M for the second time. The special node $\otimes$ cannot be in this segment, otherwise the agent A would have stopped there. If another agent B meets $\otimes$ without being blocked by M (i.e., going *CW*), then it will stop there, and when agent A or any other agent changes direction after being blocked twice, it will reach the *Stopped* agent B, before being blocked by M for the third time. So no agent reaches the fourth iteration in this case. Now suppose that an agent C meets $\otimes$ after it has been blocked by M once (i.e., going *CCW*), then agent C stops at $\otimes$. At this time, agent A must be in its third iteration (having been blocked twice by M). Agent A will be blocked by M for the third time and will reverse direction. Now agent A will reach $\otimes$ and the *Stopped* agent C before reaching M in its fourth iteration. So, if the agent A is blocked by M in its fourth iteration, this implies that no agent meets $\otimes$ in the first two iterations. Moreover, there cannot be non *Stopped* agents that are in iterations 1 and 2, since either M or A would have passed them at least once, breaking the FIFO assumption. As a consequence, all other agents have been blocked twice by M. □

We prove below the correctness of Algorithm 1. For the time complexity of the algorithm we use the concept of the *ideal time* which is the time needed if each edge traversal takes one time unit.

Lemma 2. *In any anonymous oriented ring of n nodes containing one special node $\otimes$, given $k \geq 2$ honest agents and one malicious agent, then Algorithm RV-OR solves the gathering problem of the honest agents. The time complexity is $O(n)$ ideal time steps.*

Proof. *Case 1*: No agent reaches the third iteration of the while loop. Thus no agent has been blocked by M more than once. In this case all agents will eventually reach $\otimes$ and stop there. So, gathering is achieved.
Case 2: No agent reaches the fourth iteration. Consider the first agent A to reach the third iteration of the while loop. Then this agent has been blocked twice by M (and it has changed direction twice). If this agent meets a *Stopped* agent then the *Stopped* agent must be in the special node $\otimes$. In this case any subsequent agent would also reach $\otimes$ by the same argument and thus we have gathering. Thus, either the gathering is achieved or the agent A in the third iteration does not reach a *Stopped* agent and enters the fourth iteration of the loop.
Case 3: If all agents that reach the fourth iteration, stop at $\otimes$ or at another *Stopped* agent, then we have gathering. So the only remaining case is when an agent A stops after being blocked by M in the fourth iteration. In this case, due to Lemma 1 all other agents would be blocked by M at least 2 times, i.e., they would all reach the third iteration of the loop. Since there are no *Stopped* agents, no agent stops in the third iteration and all agents eventually reach the fourth iteration. Suppose the algorithm does not achieve gathering. Then at least two agents A and B will stop at distinct locations in the fourth iteration. Note that each agent would have changed direction exactly 3 times before reaching the fourth iteration of the loop, so the agents are moving in the same direction in the fourth iteration (even though they may not reach the fourth iteration at the same time). Suppose agent A was the first agent to be blocked by M or meet $\otimes$ at the fourth iteration and to stop. At this time any other agent B is at least in the third iteration. When this agent meets either M or $\otimes$ in the third iteration, it changes its direction and now there is neither M nor $\otimes$ between this agent and agent A. So agent B will meet agent A before B reaches M or $\otimes$, thus, we will have gathering.

From Lemma 1 we know that if an agent is blocked by M in the fourth iteration of the *while* loop, then any other agent must have been blocked by M at least two times. So the worst case regarding the time complexity of achieving gathering is the one in which M is occupying $\otimes$ and all the agents move for 4 iterations. Thus, the ideal time (or time for the synchronous case) is $O(n)$ time steps. □

3.1. Improved solution for the synchronous agents

For $k \geq 2$ synchronous agents we have also devised a simpler algorithm, called Algorithm *RV-OR-IMPROVED*, that requires at most three iterations (instead of four), constant memory, and assures global termination (differently from the previous algorithm).

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* first meets an agent that is moving in the *CW* direction, let us call it *B*, *B* stops and waits, while *A* continues to move until it is blocked again by *M*. The agents that are moving in the *CW* direction and that meet *A* after *A* has met agent *B*, will just follow *A*. We use the synchronicity property to force the leader agent *A* moving *CCW* to wait at the even clocks and move at the odd clocks, vice versa the agents moving *CW* move at the even clocks and wait at the odd clocks. In this way, agents moving in opposite directions always meet at nodes. Finally, when *A* is blocked again by *M* it has collected all the remaining agents, thus reverses direction and moves back, and this is the third and last iteration. The gathering is achieved at the node of *B* and there is global termination as all the remaining agents arrive there with *A*. Finally note that, similarly to Algorithm 1, if an agent moving *CW* in the first iteration arrives in $\otimes$, stops to prevent *M* escaping forever. $\otimes$ will then become the meeting point if the leader *A* meets the first agent there while moving *CCW*, otherwise the agent in $\otimes$ will follow *A*, as *M* will anyway be blocked by the *Stopped* agent *B*.

The following theorem summarizes the results for gathering in oriented rings.

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

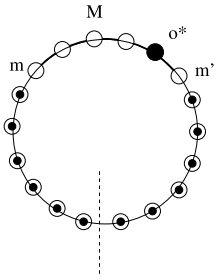


Fig. 1. An even number of agents are initially placed in the lower part of the ring between m and m', and the malicious agent can move in the remaining upper part. The agents cannot gather.

Theorem 1. *In any anonymous oriented ring of n nodes with a special node ⊗, given $k \geq 2$ honest agents and one malicious agent, then k honest agents can always solve the gathering problem even if they are asynchronous, have only constant-sized memory and have no knowledge about n or k.*

4. Impossibility results in the unoriented ring

We now consider the gathering problem with $k \geq 2$ asynchronous or synchronous honest agents in an unoriented ring consisting of n nodes with one malicious agent. We show that the problem is not solvable by asynchronous agents if k is an even number. For synchronous agents, we show that the problem is unsolvable if n is odd and k is even. All impossibility results in this section hold even if the agents have infinite memory and know the values of n and k. Note that in Section 6 we will consider the special case of $k = 2$ synchronous agents, and we will show that the solvability of gathering depends on how much memory is available to the honest mobile agents.

Lemma 3. *In any anonymous unoriented ring of n nodes containing one special node ⊗, given an even number k of honest agents initially located at distinct nodes and one malicious agent, then the gathering problem of the honest agents cannot be solved if (1) the honest agents are asynchronous, or (2) the honest agents are synchronous and n is odd. These results hold even if the agents have unlimited memory resources.*

Proof. Consider first the case n odd, and the following initial configuration fixed by an adversary: the honest agents are initially placed in k consecutive nodes and the remaining nodes, including node ⊗, are unoccupied. The malicious agent M is initially placed at ⊗. The adversary splits the chain of consecutive initially occupied nodes into two subchains of equal length and forces any two agents to have a different orientation of the ring if and only if they belong to different subchains. The configuration is symmetric with an axis of symmetry crossing the chain of agents in the middle. Each honest agent has a symmetric counterpart on the other side of the axis. Suppose there is an algorithm $\mathcal{A}$ which solves the gathering problem. Irrespective of the actions of the algorithm, the adversary can synchronize the agents so that at any given time, any two symmetric agents which initially belong to different subchains have the same input: They always arrive at symmetric nodes through the same (perceived) direction, and symmetric nodes are either unoccupied, or occupied by the same number of agents. If the agents are blocked by M, this happens at the same time. Hence the configuration remains symmetric at every time step and any two symmetric agents belonging to different subchains will never occupy the same node at the same time.

Consider now the case of n even and k even asynchronous agents, and a symmetric configuration as shown in Fig. 1. Now the adversary can prevent the asynchronous agents from approaching nodes m, m', by introducing suitable delays on the movements of the agents. Therefore, like before, the configuration will remain symmetric at every time step and any two symmetric agents belonging to different subchains will never occupy the same node at the same time. □

Note that, for n even and k even synchronous agents the above argument (for the asynchronous agents) does not hold anymore since M can protect at most one node at a given time unit.

5. Gathering in an unoriented ring

In this section we present two algorithms for solving gathering of k agents in spite of a malicious agent, in an unoriented ring network consisting of n nodes. It is not very difficult to see that the Algorithm RV-OR does not solve the problem in an unoriented ring even for feasible configurations of synchronous agents. It might for example gather the agents in more than one distinct node.

We first give an algorithm that solves the problem for asynchronous agents in all feasible configurations (i.e., when k is an odd number). We then give an algorithm that solves the problem for $k > 2$ synchronous agents in all feasible configurations (i.e., when k is an odd number, or when n is an even number). In both algorithms the agents have constant memory and do not know n or k, while all nodes of the ring are anonymous except one which is labeled as ⊗.

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

Algorithm 2 *RV-UR-ASYNC*: Gathering of asynchronous agents in unoriented rings.

```
   # Assumptions: DIR := CW;
   # Start in state: Initial

 1: if Initial then
 2:     Move(DIR);
 3:     if You meet node ⊗ unoccupied for the third time then
 4:         Become Stopped;
 5:     else if blocked by M while trying to move from a node that hosts only you then
 6:         Become Stopped;
 7:     else if You meet an agent not at node ⊗ then
 8:         if The agent you meet is alone and is a Stopped then
 9:             Become Transformer-1;
10:         else if Every other agent at the node is Final then
11:             Become Stopped;
12:         else if You meet a Stopped and at least one Final agent then
13:             Become Transformer-2;
14:         end if
15:     end if;

16: else if Transformer-1 then
17:     Wait until all other agents change to state Final;
18:     Become Searcher;

19: else if Searcher then
20:     Reverse direction (DIR := inverse(DIR));
21:     Move(DIR) until you are blocked by M while trying to move from a node u;
22:     if You see one or more agents at u and all of them are at state Final then
23:         Become Stopped;
24:     else if You see no agent at u or an agent not at state Final then
25:         Become Collector;
26:     end if

27: else if Stopped then
28:     if You see a Transformer-1 or Transformer-2 then Become Final;
29:     else if You see a Collector then Follow Collector;
30:     else if You see a Terminator then Become Terminator;
31:     end if;

32: else if Final then
33:     if You see a Collector then Become Stopped;
34:     else if You see a Terminator then Become Terminator;
35:     end if;

36: else if Collector then
37:     Wait until every other agent at the node changes its state to Stopped;
38:     Collect everyone;
39:     Reverse direction (DIR := inverse(DIR));
40:     Move(DIR) collecting every agent you meet, until you meet an agent Final;
41:     Become Terminator;

42: else if Transformer-2 then
43:     Wait until every other agent at the node changes its state to Final;
44:     Change state to Final;

45: else if Terminator then
46:     Wait until every other agent at the node changes its state to Terminator;
47:     Exit;
48: end if
```

5.1. Gathering of asynchronous agents

We now give an algorithm that solves the problem for k agents when k is odd. Intuitively, the algorithm might first gather all agents into two groups at distinct nodes. Then, one agent of the group with an odd number of agents collects all the other agents.

In our algorithm, an agent stops at $\otimes$ only when it meets $\otimes$ for the third time, while moving in the same direction. This implies that this agent has traversed the complete ring at least two times, while M has moved at least once around the ring. So, there could be no agents moving in the opposite direction. On the other hand, if the first agent that stops is one that was blocked by M, then this agent moves to the *Stopped* state and any agent moving in the same direction would reach this *Stopped* agent before reaching M or meeting $\otimes$ for the third time. When two agents gather at a node v (not $\otimes$), one of the agents called the *Searcher* reverses direction and searches for the other agents. The *Searcher* only stops when it reaches another node w (not $\otimes$) containing *Stopped* agents. If the number of agents gathered at node w is even then the

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

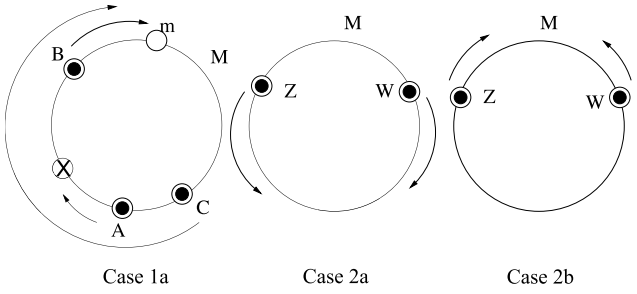


Fig. 2. The cases of Lemma 4.

Searcher becomes a *Collector* and it collects all agents and returns to node v. Note that the agent does not need to count the number of other agents as the algorithm depends only on the parity of the size of the group of agents. The agents repeatedly execute the instructions of Algorithm *RV-UR-ASYNC*.

We prove in the following lemma the correctness of Algorithm *RV-UR-ASYNC*. For the time needed for gathering we use the concept of *ideal time*, i.e., the time that would be needed if each edge traversal takes one time unit.

Lemma 4. *In any anonymous unoriented ring of n nodes containing one special node* $\otimes$*, given* $k \geq 3$ *honest asynchronous agents and one malicious agent, and given k odd, then Algorithm RV-UR-ASYNC solves the gathering problem of the honest agents. The time complexity is* $O(n)$ *ideal time steps.*

Proof. *Case 1*: All agents have the same orientation. Let A be the first agent which exits the *Initial* state (either by meeting node $\otimes$ for the third time, or by being blocked by M).

- Suppose that A meets node $\otimes$ (for the third time) and therefore A becomes a *Stopped* (see Fig. 2, case 1a). Consider the next agent B which exits the *Initial* state. Agent B should be blocked by M. Let m be the node where B is located when it tries to move and is blocked M. B becomes a *Stopped*. Then let C be the next agent that exits the *Initial* state. Agent C should meet agent B (notice that if C passes from the special node $\otimes$ will not stop to meet agent A). Thus C passes from the cases corresponding to lines 9, 16, 19, 36, 45, of the Algorithm *RV-UR-ASYNC*, collecting agent A and ending at node m. Meanwhile, all other agents (if any) gather at node m, since they pass from the case at line 10 and then possibly line 27, or line 12 and then line 42, all of them ending at state *Final* except possibly one which is at state *Stopped*. Hence, when agent C arrives at m, all the agents change to state *Terminator* and exit.
- Suppose that A is blocked by M when it tries to move from a node m. Consider the next agent B which exits the *Initial* state. If B meets node $\otimes$ then we can argue similarly as in the previous paragraph by replacing agent A there with B. If B meets A, then we can again argue as in the previous paragraph by replacing agents B and C there with A and B respectively.

Case 2: Not all agents have the same orientation. We first observe that if there are at least two agents X and Y having a different orientation for the ring, then no agent will exit the *Initial* state by stopping at the special node $\otimes$. Indeed, let $\mathcal{C}^X$ and $\mathcal{C}^Y$ be the subsets of agents having the same orientation for the ring as agents X and Y, respectively. Due to the presence of the malicious agent M in the ring (and the fact that M can not cross a honest agent), an agent from $\mathcal{C}^X$ (or $\mathcal{C}^Y$) can pass at most twice from the special node $\otimes$ and then it will be blocked by M: a) Suppose that M is before an agent $Z \in \mathcal{C}^X$ and after an agent $W \in \mathcal{C}^Y$ on the direction that Z is moving (see Fig. 2, case 2a). Then Z may pass at most twice from the special node $\otimes$ before it is blocked by M. b) Suppose that M is after an agent $Z \in \mathcal{C}^X$ and before an agent $W \in \mathcal{C}^Y$ on the direction that Z is moving (see Fig. 2, case 2b). Then Z may pass at most once from the special node $\otimes$ before it is blocked by M.

Hence let $A \in \mathcal{C}^X$ without loss of generality be the first agent which exits the *Initial* state and is blocked by M when it tries to move from a node m, and let B (which should belong to $\mathcal{C}^Y$ due to the FIFO links) be the first agent which exits the *Initial* state and it is blocked by M when it tries to move from a node m'. Both agents A and B become *Stopped*. Then let C be the next agent that exits the *Initial* state and meets A or B. Suppose without loss of generality that C meets A (and therefore $C \in \mathcal{C}^X$). Hence C becomes a *Searcher* and moves *CCW*. All other agents (if any) belonging in $\mathcal{C}^X$ will gather at node m. A similar situation may appear at node m' if there are more agents belonging in $\mathcal{C}^Y$ (i.e., having the same orientation for the ring as agent B). Nevertheless there will be at most two *Searchers* C, D and they will have different orientation for the ring. All other agents will gather at nodes m and m' before the *Searchers* D and C arrive at m and m' respectively. If the total number k of the honest agents is odd, then exactly one of the sets $\mathcal{C}^X$, $\mathcal{C}^Y$ has an odd number of agents. Suppose without loss of generality that $\mathcal{C}^X$ has an odd number of agents. Then at node m exactly one of the agents there, it will be at state *Stopped* while at node m' there will be only agents at state *Final*, before the corresponding searcher

arrives at each node. Hence only the searcher which reaches m will change its state to *Collector*, will collect all agents and gather with everyone at node m' (the other searcher which arrives at m' will stay there).

In any case above, it is easy to see that gathering occurs within $O(n)$ ideal time. □

In view of Lemmas 3, 4 the following theorem holds:

Theorem 2. *In any anonymous unoriented ring of n nodes containing one special node ⊗, given k honest asynchronous agents having constant-sized memory and no knowledge about n or k and one malicious agent, then the honest agents can solve the gathering problem if and only if k is odd.*

5.2. Gathering of synchronous agents

We now give an algorithm for all feasible configurations of $k > 2$ synchronous agents in a ring of n nodes, that is when either k is odd or n is even.

Algorithm 3, called *RV-UR-SYNC*, 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 behavior accordingly.

If an agent continues moving in the same direction and eventually reaches ⊗ 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 ⊗ twice – will stop at ⊗ and will move to state *Star*. 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. The first agent that meets the *Stopped* agent, changes to state *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 moves to state *R_Messenger*, returns back, collecting the *Star* agent on the way, and gathering 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.

More precisely, let agent A for G_1, and agent B for G_2, be the agents that are blocked by M and become *Stopped* (line 5). 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 *Messenger*, reverse direction and move towards B and A, respectively. When agents C and D meet the two *Head* agents B and A (line 64), they reverse direction and become *R_Messenger* 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 21), and gathering 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 22). 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 36).

Lemma 5. *In any anonymous unoriented ring of n nodes containing one special node ⊗, given $k > 2$ synchronous honest agents and one malicious agent, and given that k is odd or n is even, then Algorithm RV-UR-SYNC solves the gathering problem of the honest agents. The time complexity of the algorithm is $O(n)$ time steps.*

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.

Case 1: We first prove that in this case exactly one agent becomes *Stopped*. Let us first prove that at least one 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 at least one agent that becomes *Stopped*, b) or M keeps on moving in the same direction chosen by the agents, but in this case one agent will cross ⊗ twice and will stop (line 13), thus blocking M. Therefore, at least another agent will eventually be blocked by M (no other agent stops at ⊗), and will become *Stopped*. 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 *Messenger* (line 10), while A changes its state to *Head*.

Algorithm 3 *RV-UR-SYNC*: Gathering of $k > 2$ synchronous agents in unoriented rings.

```
    # Assumptions: sync = parity = star_node = 0, moved = −1, first_clock = 1, DIR := CW
    # Start in state: Initial

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

2:  if Initial then
3:      if first_clock = 1 and arrived at ⊗ then star_node := star_node + 1
4:      end if;
5:      if Blocked then Become Stopped;
6:      else
7:          Move(DIR);
8:          if arrived at ⊗ then star_node := star_node + 1
9:          end if;
10:         if Meet a Stopped agent then Become Messenger;
11:         else if Meet a Head agent h then Become Follower of h;
12:         else if Blocked then Become Stopped;
13:         else if star_node = 2 and not found a Star agent then Become Star;
14:         end if
15:     end if;

16: else if Head then
17:     if Meet a Initial agent then parity := (parity + 1) mod 2;
18:     else if Meet a R_Messenger agent r then
19:         if r.moved = −1 then Become Gathered;
20:         else if parity = 1 and r.parity = 0 then Become Wait;
21:         else if parity = 0 and r.parity = 1 then Become R_Head;
22:         else if moved = 0 and r.moved = 0 then
23:             Become R_Head;
24:             Wait (sync);
25:         end if
26:         moved := 0;
27:     else if not blocked and star_node < 2 then
28:         moved := 1;
29:         Move(DIR);
30:         if Reached ⊗ then star_node = star_node + 1;
31:         end if
32:     else Wait (1);
33:     end if

34: else if R_Head then
35:     Reverse direction (DIR := inverse(DIR));
36:     if Blocked then Become Failure;
37:     else if Meet a R_Head agent then Become Gathered;
38:     else
39:         Move(DIR);
40:         if Meet a R_Head or Wait agent then Become Gathered;
41:         end if
42:     end if;

43: else if Stopped then
44:     if Meet a Initial agent then
45:         Become Head;
46:         moved := 0;
47:         parity := (parity + 1) mod 2;
48:     else if Meet a Messenger agent m then Become Follower of m
49:     else Wait (1);
50:     end if;

51: else if Star then
52:     if Meet a R_Messenger agent r then Become Follower of r;
53:     else Wait (1);
54:     end if;

55: else if Follower of r then
56:     if r is Gathered then Become Gathered;
57:     else follow r;
58:     end if;
```

Note that, all the other agents in state *Initial* cannot become *Stopped*, but become *Follower* (line 11), 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, only agent A passed from state *Stopped*. Let us now prove that C will coordinate the gathering. C comes back in state *R_Messenger* (line 61) has the default *moved* value equal to -1. This value is used to know if the *Head* agents have moved or not. However C

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

Algorithm 3 *RV-UR-SYNC* algorithm – Continue

```
59: else if Messenger then
60:     Reverse direction (DIR := inverse(DIR));
61:     if Blocked then Become R_Messenger;
62:     else
63:         Move(DIR);
64:         if Meet a Head agent h then
65:             Become R_Messenger;
66:             moved := h.moved;
67:             parity := h.parity;
68:         else if Blocked or Meet a Stopped agent then Become R_Messenger;
69:         end if
70:     end if;

71: else if R_Messenger then
72:     Reverse direction (DIR := inverse(DIR));
73:     Move(DIR);
74:     if Meet a Head agent h then
75:         if moved = −1 then Become Gathered;
76:         else if (parity! = h.parity) or (moved = 0 and h.moved = 0) then
77:             Become Follower;
78:         else Become Messenger;
79:         end if
80:     end if;

81: else if Wait then
82:     if Meet a R_Head agent then
83:         Become Gathered;
84:         Change state of Follower agents to Gathered;
85:     else Wait (1)
86:     end if;
87: end if;

88: first_clock := 0
```

has not met a *Head* agent in the opposite direction. 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 gather. Note that, if one agent had previously stopped in $\otimes$, then it has to be in the path followed by *R_Messenger*, thus it will become a *Follower* of *R_Messenger* and will gather with the other agents. Thus, if all agents initially move in the same direction gathering is achieved.

Case 2: Let us assume that all the agents except one, called B, move in the same direction. They form a set that we call G_1. The agent M cannot escape both from B and from the agents of G_1, thus M has to block both B and another agent A of G_1. B becomes *Stopped* (line 5) but not *Head* because no other agent can reach it in state *Initial*, whereas A, becomes *Stopped* (line 5), and once it is reached by an *Initial* agent C, A becomes *Head* (line 45), and C becomes *Messenger* (line 10), changing direction and moving towards B. All the remaining agents of G_1 become *Follower* of A (line 11). When C and B meet, B becomes a *Follower* of C (line 48), and C moves in state *R_Messenger* (line 68) towards A with the default *moved* value equal to -1 (C has not met a *Head* agent). 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 agents have reached A and have become *Follower* (line 11), and since $moved = -1$ the agents terminate in state *Gathered* (lines 19, 56, and 75).

Case 3: At least two agents have an orientation in one direction, and another two in the other direction. Thus, exactly 2 agents become *Head* (line 45) and 2 agents become *Messenger* (line 10). Let us call them A and C – the *Head* and the *Messenger* on one side, and B and D, those on the other side, respectively. All the other agents become *Follower* of the *Head* they meet (line 11). C and D perform a tour in opposite directions: they reverse directions, reach B and A, respectively, and come back to A and B in state *R_Messenger* (line 68), delivering the *moved* and the *parity* information related to the other *Head* agent. Note that, *parity* is a variable used by each *Head* agent to compute the evenness/oddness of its group (is increased whenever a new *Initial* agent joins the group).

We now have to show that the two groups of agents meet and all the agents gather. First observe that, since agents are synchronous and links are FIFO D (C) meets A (B) in state *Messenger* before that C (D) comes back in state *R_Messenger* delivering the information on the parity/oddness (*parity* variable) and eventual movement towards M (*moved* variable) of the other group of agents. Also note that, when M is surrounded, A and B are not able to move anymore. We have two cases that depend on the oddness or evenness of k:

- ***k* is odd:** If k is odd, the *Head* agent of a group with $parity = 1$, remains stopped in state *Wait* (line 20), whereas the other *Head* agent, with $parity = 0$, starts moving in opposite direction in state *R_Head* (line 21) with all the other agents

of its group in state *Follower* (line 57). This group of agents will reach the other stopped group – regardless of the value n- and all agents will gather (lines 40, and 56).

- ***k* is even:** This case is more complicated and gathering can be achieved only when M has been surrounded, i.e., when the *Head* agents A and B cannot move anymore. This is possible since even if M is very fast, due to the synchronicity of the agents, M in one time step can only block either A or B. Note also that, none of the *Head* agents can meet $\otimes$ twice since M is surrounded before both of them could visit all the nodes of the ring.

Thus, we now have to prove that C and D perform a same number of turns (i.e., go back and forth from one *Head* agent to the other), and then they eventually gather, 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 (line 78), otherwise they try to gather. 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 24), and so the two groups move at an instant of time that preserves the odd distance and thus, they gather. 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* (line 36) since they did not gather.

The worst case time complexity has to be computed on the three possible settings described above. We consider the ideal time complexity. In the first two cases, i.e., in Case 1 when all the agents are moving in the same direction, or in Case 2 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 one 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 all agents gather. Globally, in these two cases $O(n)$ time steps are required in the worst case.

Case 3, where at least two pairs 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 blocks the agents as much as possible before it is surrounded. Assume the extreme case in which all agents start in neighboring positions and recall that at each round M might only block agents on one side. Thus, when the distance among the agents is x, while the *Messenger* agents are moving, the two *Head* agents conquer x positions. Thus, each *Messenger* moves at most $\sum_{i=0}^{\log(n)} 2^i = 2n-1$ time steps. Then, some few extra rounds are required to collect the final information or to fail. Thus, globally the time complexity is $O(n)$ time steps. □

6. Rendezvous of two synchronous agents in an unoriented ring

In this section we consider the only remaining case that we did not consider before: the case of two synchronous agents in an unoriented ring with one malicious agent. We have shown previously (see Lemma 3) that the problem is not solvable for $k=2$ agents (i) for any n if the agents are asynchronous, and (ii) for odd values of n if the agents are synchronous. These results hold irrespectively of the amount of internal memory provided to the agents; even agents with infinite memory cannot rendezvous in these cases. Surprisingly, for the remaining case of $k=2$ synchronous agents in a ring of even size, the solvability of rendezvous depends on the amount of memory. We consider this special case in this section and we show that in this case rendezvous is not solvable for agents having constant-sized memory, while we present a protocol for rendezvous of 2 agents having $O(\log n)$ memory.

6.1. Impossibility results for two synchronous agents

Consider the case of two synchronous agents with constant-sized memory in an unoriented ring. When n is even we have the following:

Lemma 6. *In any anonymous unoriented ring of n nodes containing one special node $\otimes$, given $k=2$ honest agents with constant-sized memory, initially located at distinct nodes, and one malicious agent, and given n even, then, the rendezvous problem cannot be solved.*

Proof. Consider the following initial configuration fixed by an adversary. The two honest agents A, B, and the malicious agent have been placed so that the *free connecting path* $fp(A,B)$ which connects the honest agents and does not include the malicious agent, consists of an odd number of edges, and does not include the special node $\otimes$. Let u, v be the nodes where A, B are respectively located and let $p(u,v) = fp(A,B)$. Suppose that the adversary forces the honest agents to have a different orientation of the ring.

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

Suppose there exists an algorithm $\mathcal{A}$ which rendezvous the two agents within finite time for any even number n. Algorithm $\mathcal{A}$ should instruct the agents to move and since the agents are identical and have the same input they will move synchronously either increasing or decreasing the length of $fp(A, B)$ by two edges ($|fp(A, B)| \geq 1$). Hence the length of path $fp(A, B)$ remains odd until one of the agents is blocked by the malicious agent M. It is clear that if the agents stop when they are blocked by M then A cannot be a correct algorithm, since M can block them (one by one) at distinct nodes. Therefore an agent that is blocked by M should: either be blocked for x time steps, or wait for y time units, or start moving in the opposite direction.[1] Notice that, the numbers x, y should be some fixed constants since the agents have only constant-sized memory.

We now prove that, if the number of nodes is n, and the initial locations u, v are selected so that $|p(u, v)| + x + 3 < n$, and $|p(u, v)| > 2x$, then the malicious agent M can always prevent the honest agents from approaching the special node $\otimes$, and can always block them so that when both agents start to move again the length of $fp(A, B) = p(u', v')$ is $|p(u', v')| = |p(u, v)|$, where u', v' are the agents' new positions.

Assume that the adversary forces the orientation so that agents start moving in opposite directions by increasing their distance. Let A be the first agent that is blocked by M while trying to move from node u towards direction d. Suppose that A in u is blocked by M a total number of x times before deciding to move away from u, i.e., in the opposite direction. The other agent B at the same time has moved an additional distance of x nodes, increasing the length of the free path between the agents to $|p(u, v)| + x$ (recall that this is possible since $|p(u, v)| + x + 3 < n$). Both agents wait for y time units (either before, between or after they have been blocked by M), and after A has stopped trying to move from u towards direction d, M blocks agent B which is blocked by M for x time units. Therefore agent B is ready to move again towards A (via the free path), a total number of $x + x + y$ time units after A was blocked. At that time their distance is again $|p(u', v')| = |p(u, v)|$ since while agent A was blocked for x time units, agent B increased the initial distance by x and while B was blocked for x time units, agent A decreased the new distance by x. Hence, their final distance is again an odd number. The only difference between this configuration and the initial one is that each agent is located x nodes away from its initial position. It is not difficult to see that in the next phases the malicious agent M can block the two agents so that they do not visit nodes not in $p(A, B)$, their distance is always at most $x + 1$ and before they synchronously start moving towards each-other they are at a distance 1. Hence $\mathcal{A}$ can not solve the problem. □

In view of Lemmas 3, 5, 6, the following theorem summarizes the results for constant-sized memory agents in unoriented rings.

Theorem 3. *In any anonymous unoriented ring of n nodes containing one special node $\otimes$, given k honest agents with constant-sized memory and no knowledge about n or k, and given one malicious agent, then honest agents can solve the gathering problem if and only if (1) k is odd or, (2) the agents are synchronous, $k > 2$ and n is even.*

6.2. Solution for two synchronous agents with non-constant-sized memory

In this section, we will exceptionally consider honest mobile agents that have $O(\log n)$ bits of memory. We briefly discuss a solution for rendezvous of $k = 2$ synchronous honest agents in an unoriented ring of size n, when n is even.

The algorithm works as follows. It uses the general idea of the *Controlled Distance* algorithm in a ring [37]. At each round i each agent first tries to move for $x_i = 2^i$ steps towards (its perceived) *CW* direction. Then the agent reverses direction to (its perceived) *CCW* direction, and repeats the same for $2 \cdot x_i$ steps. If the agent meets the special node $\otimes$, during these steps then it stops and waits for the other agent. Otherwise, if the agent neither meets the other agent nor reaches $\otimes$ during this round, then it goes to the next round. Once the chosen distance is sufficiently large (e.g. $x_i \geq n$), if the agents have not already met, then the agents would have surrounded M from either direction, by the end of this round. At that instant, the distance between the agents would be exactly $(n - 2)$, i.e., the segment of the ring between the honest agents, that does not contain M has exactly $(n - 1)$ nodes, an odd number. The agents would start moving in opposite directions from the end-points of this segment and would meet in the central node. Thus, the agents would rendezvous after at most $\log n$ rounds and at each round i, the agents need to count up to 2^i, thus requiring at most $O(\log n)$ bits of memory.

We provide below a pseudo-code of the algorithm (Algorithm 4) and a proof of correctness.

Lemma 7. *Algorithm RV-2-SYNC correctly solves rendezvous of $k = 2$ agents in any anonymous, unoriented ring of n nodes containing one special node $\otimes$, in the presence of one malicious agent when n is even. The algorithm terminates in $O(n)$ time and uses $O(\log n)$ bits of memory for each agent.*

Proof. We consider two cases that depend weather or not an agent has reached node $\otimes$.

Case 1: We assume that one of the agents reached node $\otimes$ during the execution of the algorithm. In this case, M must lie in one of the two segments of the ring joining the other agent to node $\otimes$. So, during the i-th round of the algorithm

[1] It may happen that numbers x and/or y are changing after each time an agent is blocked by M. However, those numbers may only change a constant number of times due to the constant number of agent's states. If those numbers are changing consider as x and y their maximum value in the analysis.

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

Algorithm 4 *RV-2-SYNC*: Rendezvous of 2 synchronous agents.

```
# Assumptions: count = 0, X = 2, DIR = CW
# Start in state: Initial
1: if state! = Stopped then
2:    while count < X do
3:       Move(DIR);
4:       if Reached ⊗ then
5:          Become Stopped and exit loop;
6:       end if;
7:       if met an agent then
8:          Become Rendezvous and exit loop;
9:       end if;
10:       count := count + 1;
11:   end while
12:   X := 2 ∗ X; DIR := Inverse(DIR); count := 0;
13: end if
14: if state = Stopped then
15:   repeat
16:      Wait (1);
17:   until (met an agent);
18:   Become Rendezvous;
19: end if;
```

when $X = 2^i > n$, the other agent tries to move for at least n steps in either directions and thus the agent must succeed in reaching the node $\otimes$ and thus rendezvous is achieved.

Case 2: No agent reaches the node $\otimes$ during the execution of the algorithm. This is possible only if the malicious agent M blocks a segment of the ring containing $\otimes$, preventing any agent from reaching this segment. Consider the j-th round of the algorithm when $X = 2^j \geq 2n$, and consider the first agent that is blocked by M in this round. This must happen within the first n steps of this round, so for the next n steps, this agent keeps trying to move in the same direction and M must keep blocking this agent to prevent it from reaching $\otimes$ (according to our assumption that no agent reaches $\otimes$). During these n steps: (i) if the second agent was moving in the same direction, it will meet the first agent and we have rendezvous; (ii) if the second agent was moving in the opposite direction, it must be blocked by M (to prevent it from reaching $\otimes$). Thus, at the end of this round, if the agents have not gathered, they would be on the two nodes adjacent to $\otimes$, with the malicious agent sitting on $\otimes$. Now both agents will change direction at the same time and start moving in opposite directions on the segment of length $n-2$ not containing M. Hence after exactly $n/2-1$ steps, they would meet at the central node of this segment.

The number of steps taken by the algorithm is given by the series $2+4+8+\cdots+2*n = O(n)$ and the agents need only $O(\log n)$ bits of memory for the count variable. □

7. Implementation

In this section we illustrate a proof-of-concept implementation based on the Lego Mindstorms EV3 platform,[2] which is widely adopted in robotic courses.

7.1. The implemented algorithms

The proof-of-concept implementation is developed using the algorithms for synchronous agents. In the case of the oriented ring we have used Algorithm *RV-OR-IMPROVED* presented in Section 3.1 while for the unoriented ring we have used Algorithm 3, called *RV-UR-SYNC*, defined in Section 5.2.

In this real implementation we also had to simulate the movements of the malicious agent M. Hence, we first present the strategy followed by M in oriented and unoriented rings. Since in both scenarios, no matter how M behaves, the agents will gather following respectively Algorithm *RV-OR-IMPROVED* and Algorithm *RV-UR-SYNC*, the aim of M is just to delay the gathering as much as possible.

Oriented ring The strategy of M consists of delaying as much as possible the blocking of an agent. Thus, M escapes as much as possible, but it will be eventually blocked by an agent waiting in $\otimes$. Then, after an agent A (the leader) has been blocked by M, M follows A in *CCW* direction, until it is blocked by another *Stopped* agent. At this point it stops. See Algorithm 5, called *Blocking-OR*.

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

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

Algorithm 5 *Blocking-OR*: Blocking strategy of M in oriented rings.

```
# Assumptions: DIR := CW;

repeat
    Move(DIR);
until (blocked by an agent);
repeat
    Wait (1)
until (blocked by an agent A moving CW);
Reverse direction (DIR := inverse(DIR));
repeat
    Follow agent A in direction DIR
until (blocked by an agent);
Become a Stopped agent;
```

Algorithm 6 *Blocking-UR*: Blocking strategy of M in unoriented rings.

```
if all honest agents agree on the orientation DIR = CW then
    repeat
        Move(DIR);
    until (blocked by an agent);
    repeat
        Wait (1)
    until (blocked by an agent in direction DIR);
    repeat
        Move(DIR);
    until (blocked by an agent);
else if exactly k − 1 agents agree on the orientation DIR = CW then
    repeat
        Move(DIR);
    until (blocked by an agent);
else if k is odd then
    DIR = direction in which the group of even size is moving;
    repeat
        Move(DIR);
    until (blocked by an agent);
else if k is even then
    repeat
        alternatively block a Head agent coming from one or the other direction;
    until (blocked in both directions);
end if;
Become Stopped;
```

Lemma 8. *In any anonymous oriented ring of n nodes containing one special node ⊗, given $k \geq 2$ honest agents and one malicious agent M, then the execution of Algorithm* Blocking-OR *by M causes the longest delay of the gathering of the honest agents that follow Algorithm RV-OR-IMPROVED.*

Proof. Since all honest agents follow Algorithm *RV-OR-IMPROVED*, the leader A was blocked twice by M before collecting the other agents. Thus, M tries to escape from A as long as possible but it has to stop either when it meets an agent at ⊗ or some *Stopped* agent. □

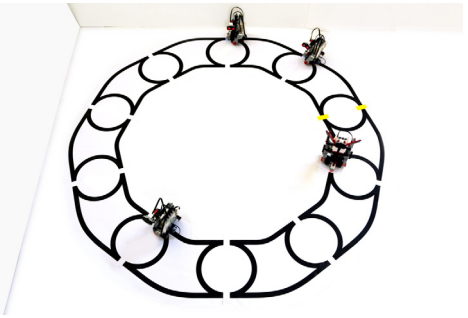
Unoriented ring If all honest agents agree on the ring's orientation, then M moves together with the agents in the same direction until it is blocked by an agent A. Then M waits until agent A moves (i.e., agent A has become a *Follower* of a *R_Messenger*), and it follows A, until it is blocked again and stops. This happens after an agent at state *Head* has crossed twice the node ⊗ and has become *Stopped*.

If exactly $k-1$ honest agents agree on the ring's orientation *CW*, then M moves *CW* until it is blocked by an agent and stops.

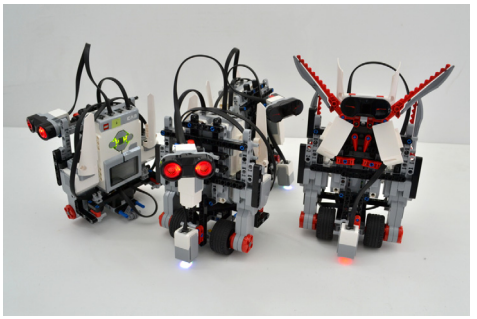
If x honest agents agree on the ring's orientation, where $2 \leq x \leq k-2$ and k is odd, then M moves towards the direction in which the group of even size is moving. If k is even then M alternatively blocks *Head* agents coming from one or the other direction, so that to maximize the number of rounds taken by the *Head* agents. See Algorithm 6, called *Blocking-UR*, for more details and also the proof of the following lemma.

Lemma 9. *In any anonymous unoriented ring of n nodes containing one special node ⊗, given $k > 2$ synchronous honest agents and one malicious agent M, and given that k is odd or n is even, then the execution of Algorithm* Blocking-UR *by M causes the longest delay of the gathering of the honest agents that follow Algorithm RV-UR-SYNC.*

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002



(a) The graph used in our experiments.



(b) Our custom Lego Mindstorms EV3 robots.

Fig. 3. The simulation setup. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Proof. Since all honest agents follow Algorithm *RV-UR-SYNC*, if all agents agree in the ring's orientation, they will move *CW* for two rounds unless they have been blocked by M earlier. Hence, M tries to delay this blocking as much as possible. Once M is blocked it waits and then it follows the agent and tries to delay the blocking of the *Head* as much as possible. Anyway, *Head* will stop after having crossed $\otimes$ twice.

If exactly $k-1$ honest agents agree on the orientation *CW*, then M should try to escape from the $k-1$ agents since one of the agents of this group will gather the other agents after moving *CCW* and *CW*. Thus, this step has to be delayed as much as possible.

If x agents agree on the orientation, where $2 \leq x \leq k-2$, and k is odd, then M tries to delay as much as possible the agent that collects all agents that belong to the group of even size. The moment of gathering depends on the time the second *Initial* agent meets a *Stopped* agent that has been blocked by M, so this time has to be maximized. The *Initial* agent, which now becomes *Messenger*, spends $n-1$ time units to reach the group of even size, and another $n-1$ time units to go back. The last case is when k is even. In this case to maximize the gathering time, M has to delay the meeting phase in the middle node, thus at each time unit it blocks one of the *Head* agents. □

7.2. Technical implementation aspects

While implementing the algorithms for the solution of the gathering problem using real honest robots and in spite of a malicious robot, in the oriented and unoriented ring networks, we had to take into account different practical aspects.

Hardware We have used standard Lego EV3 components in order to build our robots. We have used the *EV3 Color Sensor* to let the robots detect and follow the lines representing the edges of the graph. We have used the *EV3 Infrared Seeking Sensor* and *EV3 Ultrasonic Sensor* to permit the detection of other honest agents located in the same node, and to avoid collisions while following other robots. By rotating the sensors on top of the *EV3 Servo Motors* we were also able to detect robots placed crosswise. We have implemented face to face communication using the built-in Bluetooth adapters.

Software The software platform used for the implementation is *ev3dev*,[3] an open-source Linux-based operating system fully compatible with the Lego Mindstorms robots. EV3 hardware can be controlled by 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,[4] together with some exemplifying videos.[5]

The real network and the real robots To obtain a physical representation of the network we have implemented each undirected edge of the ring as two parallel links, so that real 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 their borders. Node $\otimes$ is labelled with a yellow marker. A picture of the graph used in our experiments is provided in Fig. 3a. Fig. 3b shows the honest robots which are all identical, and the malicious agent that has been physically modified to be easily identified.

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

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

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

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002

Technical limitations Even the simulation of a simple protocol that uses real robots is not straightforward. The most critical challenge we faced during the implementation of the algorithms was the agent identification. While both the Ultrasonic Sensor and the Infrared Sensor can measure the distance of objects within a limited range, the latter could be theoretically exploited to track the positions of up to four agents, each one equipped with an Infrared Beacon communicating on a specific channel. Unfortunately, our experiments showed that the Infrared Sensors are highly susceptible to interferences and to variations of the testing environment, making this approach infeasible. Due to the difficulty of relying solely on the available sensors, we managed to instrument a memory-based communication facility based on a wireless network using the built-in Bluetooth adapters and a central server. Each robot could access the states of the robots on the same node and could detect if they were blocked by the malicious agent placed in the next node by querying the server that was returning only this information. We claim that, on a bigger network the strength indicator would work more reliably and local communication could be implemented without resorting to a centralized server.

Another problem we experienced during the simulation concerns the Color Sensors. Sampling a given point in the graph with the four sensors often results in slightly different values. Additionally, even small ditches and light variations in the graph surface affect the retrieved color value. Thus, instead of relying on a single sample for the line tracking task, we applied a simple smoothing method based on the median of about ten samples.

8. Conclusion

We considered the problem of gathering a set of k mobile agents in presence of mobile transient faults, represented by a mobile malicious agent. We studied the problem in oriented and unoriented ring networks of n nodes, for synchronous and asynchronous agents and we presented algorithms that solve the problem in all feasible configurations of the agents. Except for the special case of $k = 2$ agents in unoriented rings, our algorithms can be executed by anonymous and indistinguishable agents having only constant-size memory, without any initial knowledge of k or n. For the case of $k = 2$ agents, we provided a different algorithm requiring $O(\log n)$ memory per agent and we showed that the same cannot be achieved with constant-size memory. Thus our algorithms are efficient both in terms on memory and time complexity.

Although the results in this paper are theoretical, we believe the model is not far from reality. We have attempted to show this by providing a proof-of-concept implementation of the algorithms with physical Lego Mindstorms EV3 robots in an environment closely resembling the theoretical model. The practical implementation of mobile robot algorithms is still a subject of future research and our implementation only highlights some of the challenges in this direction.

On the algorithmic side, our results completely solve the gathering problem for ring networks with synchronous or asynchronous agents. In our algorithms, we used the symmetry breaking mechanism of a landmark node that is marked as $\otimes$. Gathering in fault-free rings have been studied using other mechanisms for symmetry breaking such as tokens placed on nodes or distinct identities for agents. Such solutions require the agents to have non-constant memory (proportional to the values of n and k). If we equip the agents with non-constant memory then we could possibly extend our algorithms to work in scenarios without a landmark node, by combining the known techniques for fault-free gathering with the ideas presented in this paper. This is one direction for future work. Another interesting direction would be to study algorithms in the case of multiplicities of agents in the initial configuration (i.e. if the agents do not all start at distinct vertices of the ring). In terms of synchrony we studied fully synchronous and completely asynchronous agents; it is possible to consider other levels of synchrony between the two extremes. On the other hand one can also consider imposing synchrony of the malicious agents or limiting their speed.

Another interesting open direction is the study of more general network topologies with possibly more than one malicious agents. Clearly in a ring with more than one malicious agents, the problem cannot be solved if the malicious agents separate the ring into more than one disjoint segments, each containing some honest agents, as explained before. However, if all the malicious agents are located in a continuous segment of the ring with no honest agent in between, the algorithms we have presented for asynchronous agents would work in those cases, since this scenario is not different from that of a single fast malicious agent in the ring. The only difference would be the case of synchronous agents in unoriented rings where gathering could no longer be solved for any even number of agents (as in the asynchronous case). This is because two malicious agents can completely block a segment of the ring for the complete duration of the algorithm. Thus, the solvability of gathering in a ring is well understood even for multiple malicious agents. On the other hand, solving gathering in other network topologies seems very interesting and challenging.

Acknowledgements

Shantanu Das acknowledges support from the ANR projects MACARON (anr-13-js02-0002) and ANCOR (anr-14-CE36-0002-01), Euripides Markou acknowledges support from the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: THALIS-NTUA (MIS 379414). Riccardo Focardi, Flaminia Luccio and Marco Squarcina acknowledge support from CINI Cybersecurity National Laboratory within the project FilieraSicura: Securing the Supply Chain of Domestic Critical Infrastructures from Cyber Attacks (www.filierasicura.it) funded by CISCO Systems Inc. and Leonardo SpA.

References

[1] N. Agmon, D. Peleg, Fault-tolerant gathering algorithms for autonomous mobile robots, SIAM J. Comput. 36 (1) (2006) 56–82.
[2] S. Alpern, S. Gal, The Theory of Search Games and Rendezvous, Kluwer, 2003.
[3] L. Barriere, P. Flocchini, F.V. Fomin, P. Fraigniaud, N. Nisse, N. Santoro, D. Thilikos, Connected graph searching, Inform. and Comput. 219 (2012) 1–16.
[4] Z. Bouzid, S. Das, S. Tixeuil, Gathering of mobile robots tolerating multiple crash faults, in: International Conference on Distributed Computing Systems, ICDCS 2013, IEEE Press, Philadelphia, USA, 2013, pp. 337–346.
[5] J. Chalopin, S. Das, A. Labourel, E. Markou, Tight bounds for black hole search with scattered agents in synchronous rings, Theoret. Comput. Sci. 509 (2013) 70–85.
[6] J. Chalopin, S. Das, N. Santoro, Rendezvous of mobile agents in unknown graphs with faulty links, in: Proc. of 21st International Symposium on Distributed Computing, DISC, 2007, pp. 108–122.
[7] J. Chalopin, S. Das, P. Widmayer, Deterministic Rendezvous in Arbitrary Graphs: Overcoming Anonymity, Failures and Uncertainty, Springer, New York, 2013, pp. 175–195, Ch. 12.
[8] J. Chalopin, Y. Dieudonne, A. Labourel, A. Pelc, Rendezvous in networks in spite of delay faults, Distrib. Comput. 29 (2016) 187–205.
[9] C. Cooper, R. Klasing, T. Radzik, Locating and repairing faults in a network with mobile agents, Theoret. Comput. Sci. 411 (14–15) (2010) 1638–1647, https://doi.org/10.1016/j.tcs.2010.01.011.
[10] J. Czyzowicz, S. Dobrev, E. Kranakis, D. Krizanc, The power of tokens: rendezvous and symmetry detection for two mobile agents in a ring, in: SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science. Proceedings, Nový Smokovec, Slovakia, January 19–25, 2008, 2008, pp. 234–246.
[11] J. Czyzowicz, A. Kosowski, A. Pelc, How to meet when you forget: log-space rendezvous in arbitrary graphs, Distrib. Comput. 25 (2) (2012) 165–178, https://doi.org/10.1007/s00446-011-0141-9.
[12] J. Czyzowicz, A. Pelc, A. Labourel, How to meet asynchronously (almost) everywhere, ACM Trans. Algorithms 8 (4) (2012) 37.
[13] S. Das, R. Focardi, F. Luccio, E. Markou, D. Moro, M. Squarcina, Gathering of robots in a ring with mobile faults, in: 17th Italian Conference on Theoretical Computer Science, ICTCS 2016, Lecce, Italy, September 7–9, 2016, in: CEUR, vol. 1720, 2016, pp. 122–135.
[14] S. Das, F. Luccio, 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, Patras, Greece, September 17–18, in: LNCS, vol. 9536, Springer, 2015, pp. 211–224.
[15] S. Das, M. Mihalak, R. Sramek, E. Vicari, P. Widmayer, Rendezvous of mobile agents when tokens fail anytime, in: Proceedings of the 12th International Conference on Principles of Distributed Systems, OPODIS 2008, Luxor, Egypt, December 15–18, 2008, in: LNCS, vol. 5401, 2008, pp. 463–480.
[16] M. D'Emidio, D. Frigioni, A. Navarra, Explore and repair graphs with black holes using mobile entities, Theoret. Comput. Sci. 605 (2015) 129–145, https://doi.org/10.1016/j.tcs.2015.09.002.
[17] G. Di Stefano, A. Navarra, Optimal gathering of oblivious robots in anonymous graphs and its application on trees and rings, Distrib. Comput. 30 (2) (2017) 75–86, https://doi.org/10.1007/s00446-016-0278-7.
[18] Y. Dieudonne, A. Pelc, D. Peleg, Gathering despite mischief, ACM Trans. Algorithms 11 (1) (2014) 1–28.
[19] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Multiple agents rendezvous in a ring in spite of a black hole, in: Proceedings of the 7th International Conference on Principles of Distributed Systems, OPODIS 2003, 2003, pp. 34–46.
[20] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Mobile search for a black hole in an anonymous ring, Algorithmica 48 (1) (2007) 67–90.
[21] P. Flocchini, M.J. Huang, F.L. Luccio, Decontamination of chordal rings and tori using mobile agents, Internat. J. Found. Comput. Sci. 3 (18) (2007) 547–564.
[22] P. Flocchini, M.J. Huang, F.L. Luccio, Decontamination of hypercubes by mobile agents, Networks 3 (52) (2008) 167–178.
[23] P. Flocchini, E. Kranakis, D. Krizanc, F. Luccio, N. Santoro, C. Sawchuk, Mobile agents rendezvous when tokens fail, in: 11th International Colloquium on Structural Information and Communication Complexity, SIROCCO 2012, Smolenice Castle, Slovakia, June 21–23, in: Lecture Notes in Computer Science, vol. 3104, Springer, 2004, pp. 161–172.
[24] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, C. Sawchuk, Multiple mobile agent rendezvous in the ring, in: Proc. of Latin American Theoretical Informatics Symp., 2004, pp. 599–608.
[25] P. Flocchini, G. Prencipe, N. Santoro, Distributed Computing by Oblivious Mobile Robots, Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2012.
[26] P. Flocchini, N. Santoro, Distributed security algorithms for mobile agents, in: J. Cao, S.K. Das (Eds.), Mobile Agents in Networking and Distributed Computing, John Wiley & Sons, Inc., Hoboken, NJ, USA, 2012, pp. 41–70, Ch. 3.
[27] P. Fraigniaud, A. Pelc, Deterministic rendezvous in trees with little memory, in: Proc. of 22nd Int. Symp. on Distributed Computing, in: LNCS, vol. 5318, 2008, pp. 242–256.
[28] A. Honorat, M. Potop-Butucaru, S. Tixeuil, Gathering fat mobile robots with slim omnidirectional cameras, Theoret. Comput. Sci. 557 (2014) 1–27.
[29] L.M. Kirousis, E. Kranakis, D. Krizanc, Rendezvous with flickering tokens, Unpublished manuscript, 2005.
[30] R. Klasing, E. Markou, T. Radzik, F. Sarracco, Hardness and approximation results for black hole search in arbitrary graphs, Theoret. Comput. Sci. 384 (2–3) (2007) 201–221.
[31] E. Kranakis, D. Krizanc, E. Markou, The Mobile Agent Rendezvous Problem in the Ring, Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2010.
[32] E. Kranakis, D. Krizanc, E. Markou, Deterministic symmetric rendezvous with tokens in a synchronous torus, Discrete Appl. Math. 159 (9) (2011) 896–923, https://doi.org/10.1016/j.dam.2011.01.020.
[33] F.L. Luccio, Contiguous search problem in Sierpiński graphs, Theory Comput. Syst. 44 (2009) 186–204.
[34] G.A.D. Luna, P. Flocchini, L. Pagli, G. Prencipe, N. Santoro, G. Viglietta, Gathering in dynamic rings, in: Structural Information and Communication Complexity – 24th International Colloquium, SIROCCO 2017, Porquerolles, France, June 19–22, 2017, pp. 339–355, Revised Selected Papers.
[35] A. Pásztor, Gathering simulation of real robot swarm, Teh. Vjesn. 21 (5) (2014) 1073–1080.
[36] A. Pelc, Deterministic rendezvous in networks: a comprehensive survey, Networks 59 (3) (2012) 331–347.
[37] N. Santoro, Design and Analysis of Distributed Algorithms, Wiley Series on Parallel and Distributed Computing, vol. 56, John Wiley & Sons, 2006.
[38] Y. Yamauchi, T. Izumi, S. Kamei, Mobile agent rendezvous on a probabilistic edge evolving ring, in: Third International Conference on Networking and Computing, ICNC 2012, 2012, pp. 103–112.
[39] X. Yu, M. Yung, Agent rendezvous: a dynamic symmetry-breaking problem, in: International Colloquium on Automata, Languages, and Programming, ICALP 1996, 1996, pp. 610–621.

Please cite this article in press as: S. Das et al., Gathering of robots in a ring with mobile faults, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.05.002