Improved Periodic Data Retrieval in Asynchronous Rings with a Faulty Host*

Evangelos Bampas[1,**], Nikos Leonardos[2], Euripides Markou[3,***], Aris Pagourtzis[4,†], and Matoula Petrolia[5]

[1] LaBRI, Univ. Bordeaux, Talence, France
evangelos.bampas@labri.fr
[2] LIAFA, Univ. Paris Diderot, Paris, France
nikos.leonardos@gmail.com
[3] Department of Computer Science and Biomedical Informatics, University of Thessaly, Lamia, Greece
emarkou@ucg.gr
[4] School of Electrical and Computer Engineering, National Technical University of Athens, Greece
pagour@cs.ntua.gr
[5] LINA, University of Nantes, France
stamatina.petrolia@univ-nantes.fr

Abstract. The exploration problem has been extensively studied in unsafe networks containing malicious hosts of a highly harmful nature, called *black holes*, which completely destroy mobile agents that visit them. In a recent work, Královič and Miklík [SIROCCO 2010, LNCS 6058, pp. 157–167] considered various types of malicious host behavior in the context of the *Periodic Data Retrieval* problem in asynchronous ring networks with exactly one malicious host. In this problem, a team of initially co-located agents must report data from all safe nodes of the network to the homebase, infinitely often. The malicious host can choose whether to kill visiting agents or allow them to pass through (gray hole). In another variation of the model, the malicious host can, in addition, alter its whiteboard contents in order to deceive visiting agents. The goal

* A preliminary version of this paper appeared in the proceedings of the 21st International Colloquium on Structural Information and Communication Complexity (SIROCCO 2014), July 2014, Hida Takayama, Japan, LNCS 8576, pp. 355-370.

** This work was partially funded by the ANR project DISPLEXITY (ANR-11-BS02-014) and carried out in the frame of the "Investments for the future" programme IdEx Bordeaux–CPU (ANR-10-IDEX-03-02).

*** This research has been co-financed by 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-UOA (MIS 375891).

† This research has been co-financed by 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).

is to design a protocol for Periodic Data Retrieval using as few agents as possible.
In this paper, we present the first nontrivial lower bounds on the number of agents for Periodic Data Retrieval in asynchronous ring networks. Specifically, we show that at least 4 agents are needed when the malicious host is a gray hole, and at least 5 agents are needed when the malicious host whiteboard is unreliable. This improves the previous lower bound of 3 in both cases and answers an open question posed in the aforementioned paper.
On the positive side, we propose an optimal protocol for Periodic Data Retrieval in asynchronous rings with a gray hole, which solves the problem with only 4 agents. This improves the previous upper bound of 9 agents and settles the question of the optimal number of agents in the gray-hole case. Finally, we propose a protocol with 7 agents when the whiteboard of the malicious host is unreliable, significantly improving the previously known upper bound of 27 agents. Along the way, we set forth a detailed framework for studying networks with malicious hosts of varying capabilities.

Keywords: Distributed algorithm, Mobile agent, Periodic data retrieval, Malicious host, Gray hole, Red hole, Unreliable whiteboard

1 Introduction

In distributed mobile computing, one of the main issues is the security of both the agents that explore a network and the hosts. Various methods of protecting mobile agents against malicious nodes as well as of protecting hosts against harmful agents have been proposed (see, e.g., [19] and references therein).

In particular, the exploration problem has been extensively studied in unsafe networks which contain malicious hosts of a highly harmful nature, called *black holes*. A black hole is a node which contains a stationary process destroying all mobile agents visiting that node, without leaving any trace. In the *Black Hole Search* problem (BHS in short) the goal for the agents is to locate the black hole within finite time. More specifically, at least one agent has to survive knowing all edges leading to the black hole. The problem has been introduced by Dobrev, Flocchini, Prencipe, and Santoro in [7,10]. Since any agent visiting a black hole vanishes without leaving any trace, the location of the black hole must be deduced by some communication mechanism employed by the agents. Four such mechanisms have been proposed in the literature: a) the *whiteboard* model [5,9,10,2,16] in which there is a whiteboard at each node of the network where the agents can leave messages, b) the *pure token* model [14,1] where the agents carry tokens which they can leave at nodes, c) the *enhanced token* model [6,11,23] in which the agents can leave tokens at nodes or edges, and d) the time-out mechanism (only for synchronous networks) in which one agent explores a new node and then, after a predetermined fixed time, informs another agent who waits at a safe node [21].

In an asynchronous network, the number of nodes of the network must be known to the agents, otherwise the problem is unsolvable [10]. If the graph topology is unknown, at least $\Delta+1$ agents are needed, where Δ is the maximum node degree in the graph [9]. Furthermore, the network should be 2-connected. It is also not possible to answer the question of *whether* a black hole exists in the network. If the agents have a map of the network or at least a *sense of direction* [17,18] and can use whiteboards, then two agents with memory suffice to solve the problem. In asynchronous networks with dispersed agents (i.e., not initially located at the same node), the problem has been investigated for the ring topology [8,10] and for arbitrary networks [15,3] in the whiteboard model, while in the enhanced token model it has been studied for rings [12,13] and for some interconnected networks [23]. The problem has been also studied in synchronous networks. For a survey on BHS the reader is referred to [21].

As already mentioned, a black hole is a particular type of malicious host with a very simple behavior: killing every agent instantly without leaving any trace. In reality, a host may have many more ways to harm the agents: it may introduce fake agents, change the contents of the whiteboard, or even confuse agents by directing them to ports different from the requested ones.

In [20,22], Královič and Miklík studied how the various capabilities of a malicious host affect the solvability of exploration problems in asynchronous networks with whiteboards. They first consider networks with a malicious host (called *gray hole*) which can at any time choose whether to behave as a black-hole or as a safe node. Since the malicious behavior may never appear, the agents might not be able, in certain cases, to decide the location of the malicious host. Hence, they introduce and study the so called *Periodic Data Retrieval* problem in which, on each safe node of the network, an infinite sequence of data is generated over time and these data have to be gathered in the homebase. The goal is to design a protocol for a team of initially co-located agents so that data from every safe node are reported to the homebase, infinitely often, minimizing the total number of agents used. One agent can solve the problem in networks without malicious hosts, where the problem reduces to the *Periodic Exploration* problem (e.g., see [4] and references therein) in which the goal is to minimize the number of moves between two consecutive visits of a node. When the malicious host is a black hole, the Periodic Data Retrieval and the Periodic Exploration problem are solved by the same number of agents. As observed in [20], $n-1$ agents are sufficient for solving the Periodic Data Retrieval problem in any 2-connected network of n nodes with one malicious host when the topology is known to the agents: each of the $n-1$ agents selects a different node of the network and periodically visits all other nodes. The authors show that two agents are not sufficient to solve the problem in a ring with a gray hole and they present a protocol which solves the problem using 9 agents. They also consider a second type of malicious host which behaves as a gray hole and, in addition, can alter the contents of its whiteboard; they show that 27 agents are sufficient to solve the Periodic Data Retrieval problem in a ring, under this type of malicious host.

Our contribution. In this paper, we study and refine the model of [20]. We present the first nontrivial lower bounds on the number of agents for Periodic Data Retrieval in asynchronous rings. Specifically, we show that at least 4 agents are needed when the malicious host is a gray hole, and at least 5 agents are needed when the malicious host whiteboard is unreliable. This improves the previous lower bound of 3 agents in both cases and answers an open question posed in [20]. On the positive side, we propose an optimal protocol for Periodic Data Retrieval in asynchronous rings with a gray hole, which solves the problem with only 4 agents. This improves the previous upper bound of 9 agents and settles the question of the optimal number of agents in the gray-hole case. Finally, we propose a protocol with 7 agents when the whiteboard of the malicious host is unreliable, significantly improving the previously known upper bound of 27 agents. Along the way, we set forth a detailed framework for studying networks with malicious hosts of varying capabilities.

In order to derive the lower bounds, we make extensive use of certain configurations which the adversary can enforce in a benign execution (i.e., an execution in which the malicious host obeys the protocol), in particular 2-traversals and 3-traversals (informally, configurations in which some agent traverses an edge "with the intention" of eventually advancing one or two more edges in the same direction, respectively). We are then able to exploit the fact that we can think of the adversary as not having to commit to a particular location of the malicious host as long as the execution remains benign. For the upper bound in the case of the gray hole, we use the well known *cautious step* technique, which is also employed in [20]. However, in our case the agent marks both nodes involved in the cautious step, thus considerably reducing the number of agents that can enter the same link from the opposite direction. When the malicious host whiteboard is unreliable, we employ a natural extension of the cautious step, the *cautious double step*.

2 Preliminaries

2.1 System Model

The agents operate in a ring network where each node contains one host (we will use the terms "host" and "node" interchangeably). Each host is connected to each of its two neighbors via labeled communication ports. Each port is associated with two order-preserving queues: one for incoming agents and a second one for outgoing agents. Additionally, each host contains a whiteboard, i.e., a piece of memory that is shared among the agents present in the node at any given time, and a queue of agents who are waiting to acquire access to the whiteboard. Neighboring hosts are connected via bidirected asynchronous FIFO links, forming an undirected ring. Each host is identified by a unique label, which is recorded on its whiteboard upon initialization of the system and can be read by the agents whenever they have whiteboard access.

The agents are modeled as deterministic three-tape Turing machines: the first tape serves as the private memory of the agent, the second tape holds the label

of the port to which the agent wishes to be transferred, and the third tape holds a copy of the whiteboard of the current node, if the agent has acquired access to the whiteboard. All agents are initially located on the same node of the network, which we will call "the homebase". Each agent possesses a distinct identifier and knows the complete map of the network, which are provided on the first tape of the agent upon initialization. The only way for agents to interact with each other is through the whiteboards: they are not aware of the presence of other agents on the same node or on the same link, and they cannot exchange private messages.

Each host is responsible for removing agents from the front of its incoming queue and *executing* them, i.e., advancing each agent's state according to its transition function until the agent requests to be transferred. We assume that this happens in one atomic step, i.e., as soon as one agent A is removed from the front of an incoming queue, no other agent in that node can execute a transition before A executes its own first transition. The host is also responsible for executing the agent that is at the front of the whiteboard queue. Finally, the host is responsible for removing agents from the front of its outgoing queues and transmitting them over the link to the neighboring node (the whiteboard tape is not transmitted). The host has to perform these tasks while ensuring that no queue is neglected for an infinite amount of time. Each host is capable of executing multiple agents concurrently, but at most one of them has access to the whiteboard at any given time: the one that is at the front of the whiteboard queue. The set of states of each agent contains special states corresponding to the following actions:

1. *Request the whiteboard lock* (q_{req}): When an agent enters this state, it is inserted in the whiteboard queue. We assume that this happens atomically, i.e., any other agent who subsequently enters this state will be placed in the whiteboard queue *behind* this agent. Its execution is suspended until it reaches the front of the queue. When this happens, the host continues to execute this agent (possibly concurrently with other agents who are not accessing the whiteboard) without removing him from the whiteboard queue. Simultaneously with the transition from q_{req}, the whiteboard of the node is copied to the third tape of the agent (actually overwriting it), thus the agent obtains exclusive access to the whiteboard of the node.
2. *Release the whiteboard lock* (q_{rel}): When an agent enters this state, its whiteboard tape is copied back to the whiteboard of the node and the agent is removed from the whiteboard queue.
3. *Leave through a specified port* (q_{port}): When an agent enters this state, it is atomically inserted in the outgoing queue of the port indicated on its second tape. If the agent has not yet released the whiteboard lock, its whiteboard tape is also copied back to the whiteboard of the node and the agent is removed from the whiteboard queue.

Note that an agent actually traverses a link only when the source host decides to remove it from the outgoing queue and transmit it to the target host. Link traversal is not instantaneous. Its duration is determined by the adversary.

The system is asynchronous, meaning that any agent can be stalled for an arbitrary but finite amount of time while executing any computation or traversing any link. More specifically, only the following transitions are atomic (i.e., they are executed instantaneously): $q_{\text{port}} \to q_{\text{req}}$, $q_{\text{rel}} \to q_{\text{req}}$, $q_{\text{rel}} \to q_{\text{port}}$.

We assume that the system contains exactly one malicious host which may deviate from the system specification in several ways:

Definition 1 (Malicious behavior from the malicious host). *The malicious host in the system may choose to:*

1. Kill *any agent which is stored in any of its queues or is being executed. In this case, the agent disappears without leaving any trace, apart from what it may have already written on the whiteboards.*
2. *Operate without fairness, i.e., it can neglect one or more of its queues forever.*
3. *Transmit an agent to a node different from the one that it requested to be transmitted to, or it can transmit an agent without the agent asking for a transmission, or misreport its own node label to agents requesting it.*
4. *Execute (resp. forward) any agent in the incoming or the whiteboard (resp. outgoing) queues, without respecting the queue order.*
5. *Create and execute multiple copies of an agent at any stage.*
6. *Provide to each agent that requests access to the whiteboard an arbitrary whiteboard tape, possibly erroneous or inconsistent with the whiteboard tapes that it has provided to the other agents.*

We classify the various types of malicious host behavior in order of increasing power as follows:

Definition 2. *The malicious host is called:*

- black *hole if it kills every agent that appears in any of its queues at every time* $t \geq 0$.
- black$^+$ *hole if it kills every agent that appears in any of its queues or is being executed at every time* $t \geq t_0$*, where* $t_0 \geq 0$ *is chosen by the adversary. Until time* t_0*, which may even be equal to* $+\infty$*, it acts as a safe node.*
- gray *hole if it can choose whether to deviate (or not) from the protocol in the way described in item 1 of Definition 1 at any time* $t \geq 0$.
- gray$^+$ *hole if it can choose whether to deviate (or not) from the protocol in any of the ways described in items 1-5 of Definition 1 at any time* $t \geq 0$.
- red *hole if it can choose whether to deviate (or not) from the protocol in any of the ways described in items 1-6 of Definition 1 at any time* $t \geq 0$.

The agents do not have any information on the location of the malicious host, except from the fact that the homebase is safe. Since the agents have a map of the ring, we will assume without loss of generality that they operate in a ring with n nodes labeled with $\{1, 2, ..., n\}$ in clockwise order, with node 1 being the homebase. We denote this ring by C_n. We also assume that the two ports of each node are labeled '+' and '−', the former always leading clockwise and the latter always leading counterclockwise. Initially, each agent knows only its own

identifier and is unaware of the total number of agents in the network. An agent does not have a direct way to access another agent's identifier. The only way to accomplish this is by reading a message left on some node's whiteboard by the other agent. The identifier of a node can be accessed when an agent has access to that node's whiteboard.

2.2 Periodic Data Retrieval

We assume that every host in the system generates over time an infinite sequence of data items, all of which have to eventually reach the homebase. The agents operate in the network and their aim is to deliver the data from any safe node to the homebase infinitely often. Once an agent has acquired a chunk of data items from a host, the data may be stored at an intermediate node and possibly read by another agent before reaching the homebase. This problem is known as the *Periodic Data Retrieval* problem [20].

Definition 3. *An* instance *of the Periodic Data Retrieval problem in a ring is a tuple $\langle C_n, k, \omega, m\rangle$, where C_n is an undirected labeled and oriented ring consisting of n nodes, k is a positive integer representing the number of agents starting on the homebase which is located at node 1, $\omega \in V(C_n) \setminus \{1\}$ is the malicious host, and $m \in \{$black, black$^+$, gray, gray$^+$, red$\}$ is the maliciousness level of ω as per Definition 2.*

Definition 4. *An* execution *of an algorithm on an instance is completely determined by a sequence of choices made by the adversary. The adversary can choose which agents are activated at any given time, the speed at which agents are executed and the speed at which they perform each edge traversal, as well as any malicious behavior on the part of the malicious host. An execution $\mathcal{E}'$ is a* continuation of an execution $\mathcal{E}$ from time t_0 *if $\mathcal{E}'$ is identical to $\mathcal{E}$ up to time t_0. An execution is called* benign *if the malicious host exhibits no malicious behavior.*

During an execution, we will say that an agent is *frozen*, either on an edge or at a node, if the adversary has decided to delay the actions of that agent. If an agent is frozen at some time t, the adversary has to unfreeze it at some finite time $t' > t$.

Definition 5. *Given an execution of an algorithm, a node v is said to be* t-reported *if there exists a time $t' > t$ such that at time t' the homebase whiteboard contains all the data items that v has generated up to time t.*

Definition 6. *An algorithm $\mathcal{A}$ is* (k, m)-correct *if for every Periodic Data Retrieval instance $\mathcal{I} = \langle C_n, k, \omega, m\rangle$, for every execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$, for every node $v \in V(C_n) \setminus \{\omega\}$, and for every time t, node v is t-reported.*

Remark 1. A necessary condition for v to be t-reported is that there exist a natural number r, a sequence of (not necessarily distinct) agents $A_0, \ldots, A_r$, a

sequence of nodes $v_0, \ldots, v_r$, and an increasing sequence of times $t_0 < \cdots < t_r$, such that v_0 is v, v_r is the homebase, $t \leq t_0$, and, for each i, agent A_i visits node v_i at time t_i and node v_{i+1} at time t'_i, where $t_i < t'_i < t_{i+1}$. If ω is a red hole, then in addition we must have that $\omega \notin \{v_0, \ldots, v_r\}$.

Remark 2. A correct algorithm has to behave according to Definition 6 for every *fair* execution, in the sense that no agent can be frozen forever. In some of the proofs in Section 3, we will sometimes consider unfair executions in which some agent is frozen forever. However, these unfair executions will only serve as stepping-stones for further developing the argument and it will not be implied or assumed that a correct algorithm has to achieve Periodic Data Retrieval under these executions.

Propositions 1-3 follow directly from the definitions.

Proposition 1. *Let $\mathcal{A}$ be any algorithm. Every execution of $\mathcal{A}$ on some instance $\mathcal{I} = \langle C_n, k, \omega, m\rangle$ is also an execution of $\mathcal{A}$ on $\mathcal{I}' = \langle C_n, k, \omega, m'\rangle$, where m' is after m in the sequence (black, black$^+$, gray, gray$^+$, red).*

Proposition 2. *If an algorithm is (k, m)-correct, then it is (k, m')-correct for all m' before m in the sequence (black, black$^+$, gray, gray$^+$, red).*

Proposition 3. *Let $\mathcal{A}$ be any algorithm. Every benign execution of $\mathcal{A}$ on some instance $\mathcal{I} = \langle C_n, k, \omega, m\rangle$, where $m \in \{$black$^+$, gray, gray$^+$, red$\}$, is also a benign execution of $\mathcal{A}$ on $\mathcal{I}' = \langle C_n, k, \omega', $black$^+\rangle$, for all $\omega' \in V(C_n) \setminus \{1\}$.*

3 Lower Bounds on the Number of Agents

In this section, we give lower bounds on the number of agents required to achieve Periodic Data Retrieval in rings with gray holes (Section 3.1) and red holes (Section 3.2). We give two more definitions before presenting the results.

Remark 3 (Terminology). In what follows, the terms *traversal of an edge (u, v) by an agent A* and *agent A traverses edge (u, v)* are to be taken in the undirected sense, meaning that agent A moves either from u to v or from v to u.

Definition 7 (Waiting). *Let $\mathcal{E}$ be an execution of an algorithm $\mathcal{A}$ on instance $\mathcal{I} = \langle C_n, k, \omega, m\rangle$. Let W be a set of nodes that induces a path $C(W)$ of C_n. We say that an agent A is* waiting *on W at time t_0 under $\mathcal{E}$ if the agent is in $C(W)$ at time t_0 and, under any continuation of $\mathcal{E}$ from t_0 in which agent A does not perceive any changes in the whiteboard contents of the nodes in W (with respect to their contents at time t_0) except for those made by itself, agent A never leaves $C(W)$.*
When $W = \{v\}$, we will say that agent A is waiting on the node v. *When $W = \{u, v\}$, we will say that agent A is* waiting on the edge (u, v).

Definition 8 (ℓ-traversal). *Let $\mathcal{E}$ be an execution of $\mathcal{A}$ on $\mathcal{I} = \langle C_n, k, \omega, m\rangle$ and let $\ell \geq 1$. We say that an agent A performs an* ℓ-traversal from node v_0 at time t_0 under $\mathcal{E}$ *if all of the following hold:*

1. *Nodes $v_0, v_1, \ldots, v_\ell$ are successive on the ring and none of them is the homebase.*
2. *At time t_0, agent A just left node v_0, traversing the edge (v_0, v_1).*
3. *At time t_0, no other agent is located on nodes $v_1, \ldots, v_{\ell-1}$ or their incident edges.*
4. *Under any continuation of $\mathcal{E}$ from t_0 in which agent A is not killed and the only changes in the whiteboards of nodes $v_1, \ldots, v_{\ell-1}$ (with respect to their contents at time t_0) that are observed by agent A until it reaches node v_ℓ are the changes made by itself, agent A reaches node v_ℓ in finite time without visiting node v_0 in the meantime.*

Note that a 1-traversal is simply a traversal of an edge that is not incident to the homebase when there is no other agent traversing that edge at the same time. The next proposition directly follows from Definition 8:

Proposition 4. *If there exists an execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I} = \langle C_n, k, \omega, m\rangle$ such that properties 1–3 of Definition 8 hold and, in addition, there exists a continuation of $\mathcal{E}$ from t_0 such that agent A reaches node v_ℓ in finite time without visiting node v_0 in the meantime and no other agent traverses any of the edges (v_0, v_1) and $(v_{\ell-1}, v_\ell)$ from t_0 up to the first time when agent A reaches node v_ℓ, then agent A performs an ℓ-traversal from node v_0 at time t_0 under $\mathcal{E}$.*

3.1 Three Agents are not Enough for Gray Holes

The inexistence of $(1, black)$-correct algorithm is obvious, while the inexistence of $(2, gray)$-correct algorithm has already been demonstrated in [20]. In this section, we show that no algorithm can be $(3, gray)$-correct. We achieve this by proving that, if there existed a $(3, gray)$-correct algorithm, then the adversary would be able to force one of the agents to perform a 2-traversal (Lemma 3). However, we also prove that if any agent performs a 2-traversal while executing a $(3, gray)$-correct algorithm, then the adversary can kill all three agents (Lemma 4). This establishes that a $(3, gray)$-correct algorithm cannot exist.

Before we can prove Lemma 3, we need Definition 9, Lemmas 1 and 2, and Proposition 5 below.

Definition 9. *A* 3-converging configuration *occurs at a node v whenever there is one agent on v and two agents traversing links toward v.*

Lemma 1. *Let $\mathcal{A}$ be a $(3, gray)$-correct algorithm and let $\mathcal{I} = \langle C_n, 3, \omega, gray\rangle$. If there exists a benign execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$ under which an agent A waits on a node v which is at distance two or more from the homebase at some time t_0, then there exists a benign continuation of $\mathcal{E}$ from t_0 in which a 3-converging configuration occurs at v.*

Proof. Let A be the agent that starts waiting on v at time t_0 under $\mathcal{E}$. Note that, under any benign continuation of $\mathcal{E}$ from t_0, at least one other agent B has to visit node v. Otherwise, we could choose an instance $\mathcal{I}' = \langle C_n, 3, \omega', gray\rangle$ where

$\omega' \notin \{1, v\}$, in which, by Propositions 1 and 3, we would have a benign execution under which the safe node v would never be reported. Therefore, under $\mathcal{E}$, there must exist a time $t_1 \geq t_0$ when some agent B traverses the edge (u, v), where u is a neighbor of v, while A is still waiting on v.

We now continue $\mathcal{E}$ from t_1 in an arbitrary benign manner, except that we keep agent B frozen on the link (u, v). Let $\mathcal{E}^{(1)}$ be the resulting execution. For the sake of contradiction, suppose that, under $\mathcal{E}^{(1)}$, there does not occur a 3-converging configuration at v at any time after t_1. This implies that the third agent C never traverses any link incident to v. Consider an execution $\mathcal{E}^{(2)}$ which is identical to $\mathcal{E}^{(1)}$, except that, at time t_1, agent B is killed by u while it is in its outgoing queue. Clearly, $\mathcal{E}^{(2)}$ is a valid execution for the instance $\mathcal{I}'' = \langle C_n, 3, u, gray \rangle$. Moreover, for each node, the contents of its whiteboard under $\mathcal{E}^{(2)}$ on instance $\mathcal{I}''$ at any time t are identical to the contents of its whiteboard under $\mathcal{E}^{(1)}$ on instance $\mathcal{I}$ at time t. This implies that the remaining agent C after time t_1 will perform exactly the same actions under $\mathcal{E}^{(2)}$ on $\mathcal{I}''$ as under $\mathcal{E}^{(1)}$ on $\mathcal{I}$. In particular, C will never visit node v under $\mathcal{E}^{(2)}$ on $\mathcal{I}''$. This contradicts the correctness of $\mathcal{A}$, since the safe node v will never be reported. □

Lemma 2. *Let $\mathcal{A}$ be a $(3, gray)$-correct algorithm and let $\mathcal{I} = \langle C_n, 3, \omega, gray \rangle$. Under any benign execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$, no agent can ever wait on any node at distance two or more from the homebase.*

Proof. Suppose that there exists a benign execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$, under which some agent waits on v at time t_0, where v is at distance two or more from 1. By Lemma 1, there exists a benign continuation $\mathcal{E}^{(1)}$ of $\mathcal{E}$ from t_0 in which a 3-converging configuration occurs at v at time $t_1 \geq t_0$. By Propositions 1 and 3, $\mathcal{E}^{(1)}$ is also a benign execution on $\mathcal{I}' = \langle C_n, 3, v, gray \rangle$. Consider an execution $\mathcal{E}^{(2)}$ on $\mathcal{I}'$, which is identical to $\mathcal{E}^{(1)}$ up to time t_1, at which point all three agents are unfrozen and killed by v upon arrival. The existence of $\mathcal{E}^{(2)}$ contradicts the correctness of $\mathcal{A}$. □

Proposition 5. *Let $\mathcal{A}$ be a (k, m)-correct algorithm with $m \in \{black^+, gray, gray^+, red\}$ and let $\mathcal{I} = \langle C_n, k, \omega, m \rangle$ with $n \geq 3$. Under any benign execution of $\mathcal{A}$ on $\mathcal{I}$, for every time t_0, every node must be visited at least once by some agent after t_0.*

Proof. Suppose that there exists a time t_0 and a benign execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$ in which some node v is never visited by any agent after t_0. By Proposition 3, $\mathcal{E}$ is an execution on $\mathcal{I}' = \langle C_n, k, \omega', black^+ \rangle$, where $\omega' \notin \{1, v\}$. But then v is a safe node in $\mathcal{I}'$ and it is never visited under $\mathcal{E}$ after t_0. This contradicts the correctness of $\mathcal{A}$. □

Lemma 3. *Let $\mathcal{A}$ be a $(3,\ gray)$-correct algorithm and let $\mathcal{I} = \langle C_n, 3, \omega, gray \rangle$ with $n \geq 6$. There exists a benign execution of $\mathcal{A}$ on $\mathcal{I}$ under which some agent performs a 2-traversal.*

Proof. Let $1 \to u \to v \to w$ and $1 \to u' \to v' \to w'$ be the two paths of length three extending from the homebase in the two possible directions.

We construct the claimed benign execution as follows: As long as all the agents are confined in the nodes $\{1, u, u'\}$ and in the edges incident to the homebase, we perform an arbitrary benign execution. Whenever one agent, which we will call "stray", traverses the edge (u, v) or (u', v'), we perform an arbitrary benign execution in which all agents except the stray agent are frozen until either it goes back to distance one from the homebase (i.e., back to node u or u') or it reaches distance three from the homebase (i.e., node w or w').

Note that, in view of Lemma 2, the stray agent never waits on nodes v, v'. Therefore, every stray agent eventually either goes back to distance one or reaches distance three from the homebase. Moreover, by Proposition 5, we must have at least one stray agent and, in particular, we must have at least one that reaches distance three from the homebase. Let A be the first stray agent that reaches distance three (without loss of generality, assume that it reaches node w), and let t_0 be the last time before it reached w at which it traversed the edge (u, v).

We claim that agent A performed a 2-traversal from node u at time t_0. It is straightforward to verify that properties 1–3 of Definition 8 are satisfied. The claim follows by Corollary 4. □

Before we can prove Lemma 4, we need Proposition 8 below. We first state Propositions 6 and 7 which are directly implied by the model and the actions of the adversary.

Proposition 6. *Let $\mathcal{A}$ be any algorithm and let $\mathcal{E}$ be a benign execution of $\mathcal{A}$ on $\mathcal{I} = \langle C_n, k, \omega, m\rangle$, $m \in \{black^+, gray, gray^+, red\}$, such that at some time t_0 a set D of agents are either on a certain node $v \neq 1$ or on its incident edges, traversing them in any direction. Then, there exists an execution $\mathcal{E}'$ of $\mathcal{A}$ on $\mathcal{I}' = \langle C_n, k, v, black^+\rangle$ such that all agents in D are killed by v.*

Proposition 7. *Let $\mathcal{A}$ be any algorithm and let $\mathcal{E}$ be a benign execution of $\mathcal{A}$ on $\mathcal{I} = \langle C_n, k, \omega, m\rangle$, $m \in \{black^+, gray, gray^+, red\}$, such that after some time t_0 a set D of agents are frozen either on a certain node $v \neq 1$ or on its incident edges, while traversing them in any direction. Furthermore, no other agent traverses the edges incident to v after time t_0. Then, there exists an execution $\mathcal{E}'$ of $\mathcal{A}$ on $\mathcal{I}' = \langle C_n, k, v, black^+\rangle$ such that all agents in D are killed by v and the remaining agents perform exactly the same actions as under the execution $\mathcal{E}$ on $\mathcal{I}$.*

In both Propositions 6 and 7, the claimed execution $\mathcal{E}'$ is constructed by allowing inbound agents to arrive at the malicious host and then killing them, and by killing outbound agents before they leave the malicious host.

Proposition 8. *Let $\mathcal{A}$ be a (k, m)-correct algorithm with $m \in \{black^+, gray, gray^+, red\}$ and let $\mathcal{I} = \langle C_n, k, \omega, m\rangle$. Let $\mathcal{E}$ be a benign execution of $\mathcal{A}$ on $\mathcal{I}$ such that, at some time t_0, a set D of agents are traversing the same edge (u, v) in any direction, with $1 \notin \{u, v\}$. Then, there exists an instance $\mathcal{I}' = \langle C_n, k, \omega', black^+\rangle$ and an execution $\mathcal{E}'$ of $\mathcal{A}$ on $\mathcal{I}'$ under which at least $\min\{k, |D| + 1\}$ agents are killed.*

Proof. Let D denote the set of agents traversing edge (u, v) at time t_0 under $\mathcal{E}$. By Proposition 6, there exists an execution on $\mathcal{I}' = \langle C_n, k, v, black^+ \rangle$ under which at least $|D|$ agents are killed. This proves the claim in the case where $|D| = k$. Now, assume that $|D| < k$. Consider an arbitrary benign continuation $\mathcal{E}'$ of $\mathcal{E}$ from t_0 in which all the agents in D are frozen on the edge (u, v). At least one agent must visit node u or v under $\mathcal{E}'$ after time t_0.

If not, then we obtain a contradiction as follows: By Proposition 3, $\mathcal{E}'$ is a benign execution of $\mathcal{A}$ on $\mathcal{I}' = \langle C_n, k, v, black^+ \rangle$. Moreover, by Proposition 7, there exists an execution $\mathcal{E}''$ of $\mathcal{A}$ on $\mathcal{I}'$ under which all agents in D are killed and, by assumption, no agent ever visits the safe node u after time t_0. Therefore, algorithm $\mathcal{A}$ is not $(k,\ black^+)$-correct and hence, by Proposition 2, it is not (k, m)-correct for any $m \in \{\text{black}^+, \text{gray}, \text{gray}^+, \text{red}\}$.

So, without loss of generality, some agent $A \notin D$ visits node u at some time after t_0 under execution $\mathcal{E}'$ on $\mathcal{I}$. By Proposition 6, there exists an execution on $\mathcal{I}'' = \langle C_n, k, u, m \rangle$ under which all the agents in $D \cup \{A\}$ are killed by u. □

Lemma 4. *Let $\mathcal{A}$ be a $(3,\ gray)$-correct algorithm and let $\mathcal{I} = \langle C_n, 3, \omega, gray \rangle$. Under any benign execution of $\mathcal{A}$ on $\mathcal{I}$, no agent can ever perform a 2-traversal.*

Proof. Suppose that there exists a benign execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$, under which some agent A performs a 2-traversal from v at time t_0, and let v, w, x be the nodes that A intends to traverse. We continue $\mathcal{E}$ from t_0 by performing an arbitrary benign execution in which agent A is frozen on (v, w). Let $\mathcal{E}^{(1)}$ be the resulting benign execution. Under $\mathcal{E}^{(1)}$, there must exist a time $t_1 \geq t_0$ when another agent B traverses a link incident to w.

If not, then we obtain a contradiction as follows: Let $\mathcal{E}^{(2)}$ be an execution which is identical to $\mathcal{E}^{(1)}$, except that at time t_0, agent A is killed by node v while it is in its outgoing queue. Clearly, $\mathcal{E}^{(2)}$ is a valid execution on instance $\mathcal{I}' = \langle C_n, 3, v, gray \rangle$, under which the safe node w is never reported after t_0.

Now, under $\mathcal{E}^{(1)}$, agent B traverses one of the two links (v, w) or (x, w) at time t_1. In the first case, Proposition 8 applies immediately and it yields an instance on which all three agents are killed, thus contradicting the correctness of $\mathcal{A}$. In the second case, we continue $\mathcal{E}^{(1)}$ from t_1 by freezing all agents except A. Since A is in the process of performing a 2-traversal and the whiteboard of w has not changed with respect to time t_0, A will traverse the link (w, x) in finite time. Again by Proposition 8, we get that $\mathcal{A}$ is incorrect. □

By Lemmas 3 and 4, the existence of a $(3, gray)$-correct algorithm yields a contradiction. Therefore, we have proved the following:

Theorem 1. *There does not exist a $(3, gray)$-correct algorithm.*

Remark 4. We have stated the lemmas in this section assuming the existence of a $(3, gray)$-correct algorithm $\mathcal{A}$. However, a careful examination of the proofs of Lemmas 1-4 reveals that the proofs go through even under the weaker assumption of the existence of a $(3, black^+)$-correct algorithm. Consequently, Theorem 1 is, in fact, a corollary of the following stronger theorem:

Theorem 2. *There does not exist a $(3, black^+)$-correct algorithm.*

3.2 Four Agents are not Enough for Red Holes

In view of Proposition 2, the impossibility result in [20] together with Theorem 1 imply that there do not exist $(1, red)$-correct, $(2, red)$-correct, or $(3, red)$-correct algorithms. In this section, we show that no algorithm can be $(4, red)$-correct. To this end, we first prove in Lemma 7 that, under any $(4, red)$-correct algorithm, the adversary can force some agent to perform a 3-traversal (in fact, this can even be enforced under any $(4, gray)$-correct algorithm). Then, we derive a contradiction by showing in Lemma 8 that if an agent performs a 3-traversal, then four agents can die in the red hole and thus the algorithm cannot be $(4, red)$-correct.

Before proving Lemma 7, we need Lemmas 5 and 6 below.

Lemma 5. *Let $\mathcal{A}$ be a $(4, gray)$-correct algorithm and let $\mathcal{I} = \langle C_n, 4, \omega, gray\rangle$ with $n \geq 9$. If there exists a benign execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$ under which an agent waits on a node which is at distance two or more from the homebase at some time t_0, then there exists a benign continuation of $\mathcal{E}$ from t_0 in which some agent performs a 3-traversal.*

Proof. We will construct a continuation of $\mathcal{E}$ from t_0 with the desired property. Let A be the agent that starts waiting on some node v at time t_0. Note that, under any benign continuation of $\mathcal{E}$ from t_0, at least one other agent B has to visit node v. Otherwise, we could choose an instance $\mathcal{I}' = \langle C_n, 4, \omega', gray\rangle$ with $\omega' \neq v$, in which, by Propositions 1 and 3, we would have a benign execution under which the safe node v would never be reported. Therefore, under $\mathcal{E}$, there must exist a time $t_1 \geq t_0$ when some agent B traverses the edge (u, v), where u is a neighbor of v.

We now continue the execution from t_1 in an arbitrary benign manner, except that we freeze every agent that traverses edge (u, v). Let $\mathcal{E}^{(1)}$ be the resulting execution and let w be the other neighbor of v. Exactly one of the following can happen under $\mathcal{E}^{(1)}$:

1. *No agent ever traverses edge (w, v) after t_1 and there exists a time $t_2 \geq t_1$ when $k \geq 1$ agents are frozen on edge (u, v) and no agent ever attempts to traverse edge (u, v) after t_2.*
2. *There exists a time $t_2 \geq t_1$ when some agent C traverses edge (w, v) and $k \geq 1$ agents are frozen on edge (u, v).*

In case 1, consider an execution $\mathcal{E}^{(2)}$ which is identical to $\mathcal{E}^{(1)}$ except that, at time t_2, all agents which are attempting to traverse edge (u, v) are killed by u while they are in its outgoing queue. Clearly, $\mathcal{E}^{(2)}$ is a valid execution for the instance $\mathcal{I}' = \langle C_n, 4, u, gray\rangle$. Moreover, for each node, the contents of its whiteboard under $\mathcal{E}^{(2)}$ on instance $\mathcal{I}'$ at any time t are identical to the contents of its whiteboard under $\mathcal{E}^{(1)}$ on instance $\mathcal{I}$ at time t. This implies that any remaining agents after time t_2 will perform exactly the same actions under $\mathcal{E}^{(2)}$ on $\mathcal{I}'$ as under $\mathcal{E}^{(1)}$ on $\mathcal{I}$. In particular, no agent will traverse the edge (w, v) under $\mathcal{E}^{(2)}$ on $\mathcal{I}'$. This contradicts the correctness of algorithm $\mathcal{A}$, since the safe node v will never be reported. Thus, case 1 cannot happen.

In case 2, note that we must have $k \leq 2$, since there are a total of four agents operating on C_n. We can rule out the case $k = 2$ as follows: $\mathcal{E}^{(1)}$ is a benign execution, so by Propositions 1 and 3 it is valid for instance $\mathcal{I}'' = \langle C_n, 4, v, gray \rangle$. But, on instance $\mathcal{I}''$, we can continue $\mathcal{E}^{(1)}$ from t_2 by unfreezing all agents and killing all of them on node v. This contradicts the correctness of $\mathcal{A}$.

The only remaining possibility, then, is to have agent A waiting on node v, agent B frozen on edge (u, v), and agent C attempting to traverse edge (w, v) at time t_2 under $\mathcal{E}^{(1)}$. We freeze agent C at time t_2 and continue $\mathcal{E}^{(1)}$ from t_2 in an arbitrary benign manner while keeping agents B, C frozen. Let $\mathcal{E}^{(3)}$ be the resulting execution. By reiterating the argument of the previous paragraph, we can rule out the possibility of the remaining agent D traversing any of the edges (u, v), (w, v). We now claim that, while agents B and C are frozen, agent D must visit periodically all of the nodes in C_n except v. Indeed, if there is a time $t_3 \geq t_2$ after which agent D never visits node $x \neq v$ under $\mathcal{E}^{(3)}$, then consider the execution $\mathcal{E}^{(4)}$ on $\mathcal{I}''$ which is identical to $\mathcal{E}^{(3)}$ up to time t_3, when agents B, C are unfrozen and killed together with agent A on node u. The trajectory of D under $\mathcal{E}^{(4)}$ on $\mathcal{I}''$ will be exactly the same as under $\mathcal{E}^{(3)}$ on $\mathcal{I}$, therefore the safe node x will never be reported. This contradicts the correctness of $\mathcal{A}$.

We conclude, then, that, under the benign execution $\mathcal{E}^{(3)}$, which is a continuation of $\mathcal{E}$ from t_0, agent D must traverse at least once after time t_2 the path $u \rightsquigarrow w$ that does not contain v. Moreover, agent D will be the only agent that is moving during that time. Therefore, since the ring contains at least 9 nodes, agent D will perform a 3-traversal by Proposition 4. □

Lemma 6. *Let $\mathcal{A}$ be a $(4, gray)$-correct algorithm and let $\mathcal{I} = \langle C_n, 4, \omega, gray \rangle$ with $n \geq 9$. If there exists a benign execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$ in which, at some time t_0, two agents are traversing an edge (u, v) in any direction, where $1 \notin \{u, v\}$, then there exists a benign continuation of $\mathcal{E}$ from t_0 in which some agent performs a 3-traversal.*

Proof. Let A and B be the agents that are traversing edge (u, v) at time t_0 under $\mathcal{E}$. Let $\mathcal{E}^{(1)}$ be an arbitrary benign continuation of $\mathcal{E}$ from t_0, under which agents A and B are frozen on edge (u, v).

We claim that there must exist a time $t_1 \geq t_0$ when some agent C traverses either (w, u) or (x, v) under $\mathcal{E}^{(1)}$, where w and x are the other neighbors of u and v, respectively. Otherwise, consider an execution $\mathcal{E}^{(2)}$ which is identical to $\mathcal{E}^{(1)}$ except that, at time t_0, agents A and B are killed by u. Clearly, $\mathcal{E}^{(2)}$ is a valid execution for the instance $\mathcal{I}' = \langle C_n, 4, u, gray \rangle$. Moreover, for each node, the contents of its whiteboard under $\mathcal{E}^{(2)}$ on $\mathcal{I}'$ at any time t are identical to the contents of its whiteboard under $\mathcal{E}^{(1)}$ on $\mathcal{I}$ at time t. This implies that the safe node v in $\mathcal{I}'$ will never be visited after time t_0 under $\mathcal{E}^{(2)}$. This contradicts the correctness of $\mathcal{A}$.

Without loss of generality, we assume that at time t_1, agent C traverses edge (w, u) under $\mathcal{E}^{(1)}$. Consider an arbitrary benign continuation $\mathcal{E}^{(3)}$ of $\mathcal{E}^{(1)}$ from t_1 in which agents A, B, and C are frozen. Agent D cannot traverse any

of (w, u) or (v, u), otherwise the algorithm would fail on the instance $\mathcal{I}'$ (every agent would be killed by u). Moreover, agent D must visit periodically all nodes in C_n except u, otherwise the algorithm would again fail on the instance $\mathcal{I}'$. We conclude, then, that, under the benign execution $\mathcal{E}^{(3)}$, which is a continuation of $\mathcal{E}$ from t_0, agent D must traverse at least once after time t_2 the path $w \rightsquigarrow v$ that does not contain u. Since the ring contains at least 9 nodes, agent D will perform a 3-traversal. □

Lemma 7. *Let $\mathcal{A}$ be a $(4, gray)$-correct algorithm and let $\mathcal{I} = \langle C_n, 4, \omega, gray\rangle$ with $n \geq 9$. There exists a benign execution of $\mathcal{A}$ on $\mathcal{I}$ under which some agent performs a 3-traversal.*

Proof. Let $1 \to u \to v \to w$ and $1 \to u' \to v' \to w'$ be the two paths of length three extending from the homebase in the two possible directions. Our first goal is to prove that there exists an execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$ under which some agent performs a 2-traversal.

We construct $\mathcal{E}$ as follows: We perform an arbitrary benign execution as long as all the agents are confined in the nodes $\{1, u, u'\}$ and in the edges incident to the homebase. Whenever one agent, which we will call "stray", traverses the edge (u, v) or (u', v'), we perform an arbitrary benign execution in which all agents except the stray agent are frozen until either it goes back to distance one from the homebase (i.e., node u or u'), or it reaches distance three from the homebase (i.e., node w or w').

Note that, in view of Lemma 5, we may assume that the stray agent does not wait on node v or v', for otherwise the proof is complete. Therefore, every stray agent eventually either goes back to distance one or reaches distance three from the homebase. Moreover, by Proposition 5, we must have at least one stray agent and, in particular, we must eventually have one that reaches distance three from the homebase.

Let A be the first stray agent that reaches distance three and, without loss of generality, assume that it has strayed in the direction of nodes $\{u, v, w\}$. Let t_0 be the last time before it reached w at which it traversed the edge (u, v). We claim that agent A performed a 2-traversal from node u at time t_0. Indeed, properties 1-3 of Definition 8 are easily seen to hold and the claim follows by Proposition 4.

Now, let $\mathcal{E}^{(1)}$ be an arbitrary benign continuation of $\mathcal{E}$ from t_0 in which agent A is frozen on the edge (u, v) and the other agents are unfrozen. Under $\mathcal{E}^{(1)}$, there must exist a time $t_1 \geq t_0$ when another agent B traverses a link incident to v.

If not, then we obtain a contradiction as follows: Let $\mathcal{E}^{(2)}$ be an execution which is identical to $\mathcal{E}^{(1)}$, except that at time t_0, agent A is killed by node u while it is in its outgoing queue (if A was going from u to v) or in its incoming queue (if A was going from v to u). Clearly, $\mathcal{E}^{(2)}$ is a valid execution on instance $\mathcal{I}' = \langle C_n, 4, u, gray\rangle$, under which the safe node v is never reported after t_0.

Now, under $\mathcal{E}^{(1)}$, agent B traverses one of the two links (u, v) or (w, v) at time t_1. In the first case, Lemma 6 applies immediately and it yields a benign

continuation under which some agent preforms a 3-traversal. In the second case, we continue $\mathcal{E}^{(1)}$ from t_1 by freezing all agents except A. Since A is in the process of performing a 2-traversal and the whiteboard of v has not changed with respect to time t_0, A will traverse the link (v, w) in finite time. Again by Lemma 6, we obtain a benign continuation under which some agent performs a 3-traversal. □

Lemma 8. *Let $\mathcal{A}$ be a $(4, red)$-correct algorithm and let $\mathcal{I} = \langle C_n, 4, \omega, red\rangle$. Under any benign execution of $\mathcal{A}$ on $\mathcal{I}$, no agent performs a 3-traversal.*

Proof. Suppose that there exists a benign execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$, under which some agent A performs a 3-traversal from u at time t_0, and let u, v, w, x be the nodes that A intends to traverse in that 3-traversal. Let y be the other neighbor of x.

We freeze A and perform a benign continuation of $\mathcal{E}$ from t_0 with the following properties: All agents except those which are frozen on (u, v) are executed in an arbitrary benign manner. Whenever any agent attempts to traverse (u, v), it is frozen and remains frozen. Whenever one agent, which we will call "inbound", traverses (w, x), all agents except the inbound agent are frozen. We have the following cases regarding the behavior of the inbound agent:

1. The inbound agent reaches node v in finite time, without attempting to traverse (x, y).
2. The inbound agent traverses the edge (w, x) back and forth a finite number of times without attempting to traverse (x, y), and in finite time starts waiting on w.
3. The inbound agent traverses the edge (w, x) back and forth a finite number of times without attempting to traverse (x, y), and at some finite time t_1 starts waiting on (w, x). Moreover, there exists a benign continuation from t_1 in which all agents except those on edge (u, v) are activated, such that at some time $t_2 \geq t_1$ some agent (other than the inbound agent) traverses (w, x) while the inbound agent is waiting on (w, x).
4. The inbound agent traverses the edge (w, x) back and forth a finite number of times without attempting to traverse (x, y), and at some finite time t_1 starts waiting on (w, x). Moreover, under every benign continuation from t_1 in which all agents except those on edge (u, v) are activated, no agent other than the inbound agent traverses (w, x) while the inbound agent is waiting on (w, x).
5. The inbound agent traverses the edge (w, x) back and forth a finite number of times without attempting to traverse (x, y), and in finite time starts waiting on x.
6. The inbound agent traverses the edge (w, x) back and forth a finite number of times and in finite time traverses (x, y).

Now, for each of the cases 4-6, we explain how to continue the execution so as to force the inbound agent to leave the area defined by the nodes $\{u, v\}$ and their adjacent edges and, subsequently, to have a new occurrence of an inbound agent. Next, we argue that it cannot be the case that *all* inbound agents that

occur under this scheme fall in cases 4-6. Therefore, at some point, there has to be an inbound agent that falls in cases 1-3. To conclude the proof, starting from the time of appearance of the first such inbound agent, we give two continuations in which the malicious host is w in one and x in the other, but which cannot be distinguished by the agents, and we obtain a contradiction.

If the inbound agent falls in one of the cases 4-6, then the execution continues as follows: from t_1 by performing an arbitrary benign execution of all agents except those on edge (u, v), which are frozen. There must exist a time $t_2 \geq t_1$ at which the whiteboard contents of x are such that the inbound agent stops waiting on (w, x), otherwise the algorithm fails on the instance $\mathcal{I}' = \langle C_n, 4, u, red \rangle$ by killing all agents attempting to traverse (u, v), because the safe node v will never be visited after t_0. At time t_2, we freeze every agent except the inbound agent and allow it to arrive at node x. Then, we continue the execution in an arbitrary benign manner, still keeping the agents on (u, v) frozen, until there is a new inbound agent. In cases 5 and 6, let t_1 be the time at which the inbound agent starts waiting on x or attempts to traverse (x, y). Again, we continue the execution from t_1 in an arbitrary benign manner, keeping the agents on (u, v) frozen, until there is a new inbound agent. In all three cases, there has to be a new inbound agent, otherwise the algorithm fails on instance $\mathcal{I}'$.

Our next claim is that at least one inbound agent will fall in one of the cases 1-3. Indeed, if all inbound agents fall in cases 4-6, then the algorithm fails on the instance $\mathcal{I}' = \langle C_n, 4, u, red \rangle$ by killing all agents attempting to traverse (u, v), because the safe node v will never be visited after t_0. Let B be the first inbound agent that falls in one of the cases 1-3 and let t_3 be the time at which it traverses (w, x) for the first time. Figure 1 illustrates the rest of the proof when B falls in case 1.

Let $\mathcal{E}^{(1)}$ be the benign execution described so far. We construct a continuation $\mathcal{E}^{(2)}$ of $\mathcal{E}^{(1)}$ from t_3, distinguishing three cases according to the case in which the inbound agent B belongs:

Case 1. Freeze all agents but B and activate it until the time $t_4 \geq t_3$ at which it traverses (w, x) for the last time before it moves towards v. At time t_4, freeze B and all agents on (u, v) and take an arbitrary benign continuation from t_4. At some time $t_5 \geq t_4$, some agent C must traverse (w, x), otherwise the algorithm fails on the instance $\mathcal{I}'' = \langle C_n, 4, v, red \rangle$ (the safe node w is never visited).

Case 2. Freeze all agents but B and activate it until the time $t_4 \geq t_3$ at which it traverses (w, x) for the last time before it starts waiting on w. At time t_4, freeze B and all agents on (u, v) and take an arbitrary benign continuation from t_4. At some time $t_5 \geq t_4$, some agent C must traverse (w, x), otherwise the algorithm fails on the instance $\mathcal{I}'' = \langle C_n, 4, v, red \rangle$ (B never leaves the safe node w, which is never reported).

Case 3. Freeze all agents but B and activate it until the time $t_4 \geq t_3$ at which it starts waiting on (w, x). Now, take a benign continuation from t_4 in which all agents except those on edge (u, v) are activated, and in which at some time $t_5 \geq t_4$ some agent C traverses (w, x), tweaking it a little bit if necessary so

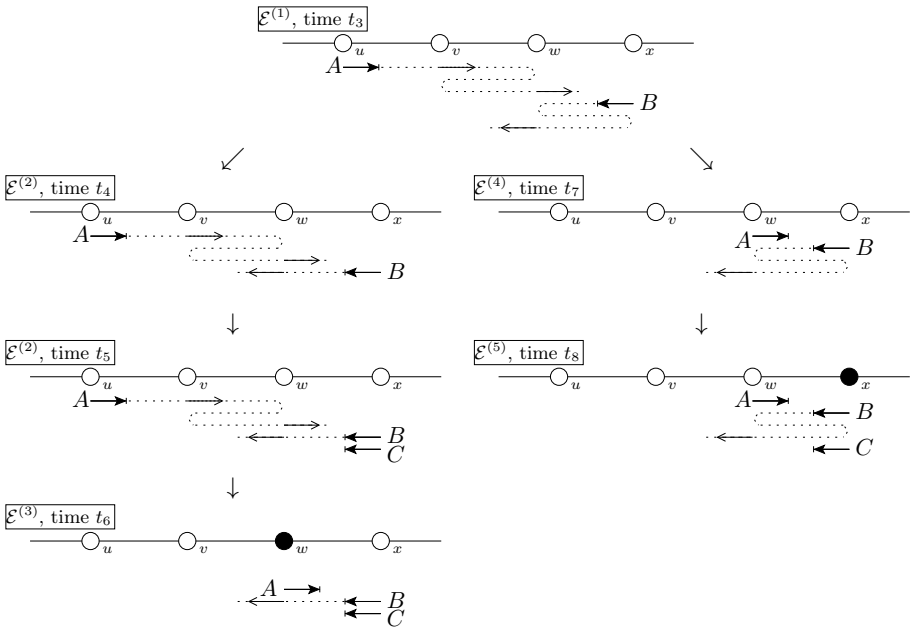

Fig. 1. Different continuations in the proof of Lemma 8, when the inbound agent B falls in case 1. Dashed lines illustrate the agent's intended trajectory. Hosts are shown as white-filled circles, except after the adversary has committed to the location of the malicious host, which is then shown as a black-filled circle.

that agent B is traversing (w, x) in any direction at time t_5. This continuation exists by our assumption for case 3.

Note that, in any case, $\mathcal{E}^{(2)}$ is a benign execution of $\mathcal{A}$ on $\mathcal{I}$, so by Propositions 1 and 3 it is a benign execution of $\mathcal{A}$ on $\mathcal{I}_w = \langle C_n, 4, w, red\rangle$. Moreover, note that the whiteboard contents of v at time t_5 are the same as at time t_0, whereas the whiteboard contents of w may have changed. Let D be the fourth agent. Consider the following continuation $\mathcal{E}^{(3)}$ of $\mathcal{E}^{(2)}$ on $\mathcal{I}_w$: Agents B, C, and D are frozen and agent A is allowed to move. Every time A visits w, the malicious node w presents to A the appropriate whiteboard contents that will allow it to conclude its intended 3-traversal by traversing the edge (w, x) at some time $t_6 \geq t_5$.

We now construct a different continuation $\mathcal{E}^{(4)}$ of $\mathcal{E}^{(1)}$ from t_3 as follows: All agents except A are frozen and A is allowed to complete its intended 3-traversal by traversing (w, x) at some time $t_7 \geq t_3$. Note that the whiteboard contents of v and w at time t_3 are the same as at time t_0, therefore the 3-traversal will indeed be completed. Note that $\mathcal{E}^{(4)}$ is a benign execution of $\mathcal{A}$ on $\mathcal{I}$, so by Propositions 1 and 3 it is a benign execution of $\mathcal{A}$ on $\mathcal{I}_x = \langle C_n, 4, x, red\rangle$. Consider the following continuation $\mathcal{E}^{(5)}$ of $\mathcal{E}^{(4)}$ from t_7 on $\mathcal{I}_x$: Agents A and B are frozen and the two remaining agents are executed exactly in the same manner as they were executed under $\mathcal{E}^{(2)}$ from time t_3 and on. Every time one or both of them visit the malicious node x, they are presented with the appropriate whiteboard contents that will allow them to replicate the execution $\mathcal{E}^{(2)}$. By construction of $\mathcal{E}^{(2)}$, at some time $t_8 \geq t_3$, the same agent C as under the execution $\mathcal{E}^{(2)}$ will traverse (w, x). Moreover, at that time, the remaining agent D will be in exactly the same position and state, and the whiteboard contents of all nodes in the path $v \rightsquigarrow 1 \rightsquigarrow y$ will be exactly the same.

This implies that, under any continuation of $\mathcal{E}^{(3)}$ from t_6 on $\mathcal{I}_x$ in which only D is allowed to move and under any continuation of $\mathcal{E}^{(5)}$ from t_8 on $\mathcal{I}_w$ in which only D is allowed to move, D will perform exactly the same actions. In particular, it must visit at least one of the nodes $\{w, x\}$, otherwise the algorithm fails on both $\mathcal{I}_w$ and $\mathcal{I}_x$ by having the respective malicious host kill the three agents A, B, and C while they are traversing (w, x). But if D visits w, then the algorithm fails on $\mathcal{I}_w$ by having the malicious host w kill all agents when D arrives. Similarly, if D visits x, the algorithm fails on $\mathcal{I}_x$. This contradicts the correctness of $\mathcal{A}$. □

By Lemmas 7 and 8 and Propositions 1 and 2, the existence of a $(4, red)$-correct algorithm yields a contradiction. Therefore, we have proved the following:

Theorem 3. *There does not exist a* $(4, red)$*-correct algorithm.*

4 An Optimal Algorithm for Rings with a Gray$^+$ Hole

In view of Theorem 1, no algorithm can achieve Periodic Data Retrieval on a ring with a gray$^+$ hole using only three agents (in fact, not even on a ring with a gray hole). In this section, we present algorithm PDR_Rings_gray$^+$, which

solves the problem in the presence of a gray$^+$ hole in a ring, using an optimal number of four agents.

Remark 5. In order to simplify the presentation, we will not make explicit the part of the algorithm that is responsible for picking up the data from nodes and delivering it to the homebase or to an intermediate node to be picked up by another agent. We assume that each agent, after getting access to the whiteboard of any node, reads all the node data that has been generated from the node or left there from other agents and also leaves a copy of the node data that it is already carrying but is not present in the node. In the following, we will deal explicitly only with the part of the algorithm that ensures that four agents are sufficient to ensure Periodic Data Retrieval in the presence of a gray$^+$ hole.

Before presenting the algorithm, we outline the interface exposed by the nodes to visiting agents:

- Each node exposes to the agents two functions: *getNodeID*() and *transfer*(*port*). The former returns the ID of the current node (recall that this may be misreported by ω). The latter places the agent in the outgoing queue of the port specified in its argument, releasing the whiteboard lock.
- Additionally, each node exposes to the agents which it is executing the whiteboard object *WB*, which has two members, *WB.list* and *WB.flags*, and two methods, *WB.access*() and *WB.release*(). *WB.access*() requests the whiteboard lock and thus results in the agent being placed in the whiteboard queue. The agent remains inactive until it reaches the front of the queue. At that point, it gains access to *WB.list* and *WB.flags*. *WB.list* contains quadruples of the form $\langle \mathrm{id}, \mathrm{op}, \mathrm{port}, s\rangle$, where id is an agent identifier, op is one of the constants $\{\mathrm{ARR}, \mathrm{DEP}\}$, port is a port number, and s is a non-negative integer. If op = ARR, the entry means that the agent with the specified id arrived from the specified port after traversing a total number of s edges from the beginning. If op = DEP, the entry means that the agent with the specified id departed from the specified port before traversing its $(s+1)$-st edge. *WB.flags* contains pairs of the form $\langle \mathrm{id}, \mathrm{dir}\rangle$, where id is an agent identifier and dir $\in \{+, -\}$. The meaning of an entry in *WB.flags* will become apparent when we describe the algorithm.

While moving from node to node, agents perform several low-level operations outlined below:

- When arriving at a node, the agent requests whiteboard access and, when this is granted, it inserts a quadruple $\langle \mathrm{id}, \mathrm{ARR}, p, s\rangle$ into *WB.list*. The agent releases the whiteboard lock just before it leaves the node, after inserting a quadruple $\langle \mathrm{id}, \mathrm{DEP}, p', s\rangle$ into *WB.list*. However, if the agent is granted whiteboard access and it detects that some other agent has inserted its ARR-quadruple but not the corresponding DEP-quadruple, it releases the whiteboard lock without writing anything and requests whiteboard access *de novo*, waiting for the other agent to conclude its computation on the node (since in

our algorithm an agent should first insert the corresponding DEP-quadruple and then release the whiteboard lock, if this does not happen, then this node must be the malicious node).
- Additionally, before leaving each node, the agent keeps a copy of *WB.list*. When arriving at the destination node, after being granted whiteboard access for the first time, the agent checks the following conditions and halts if any of them is true: *(a)* The current node, as reported by *getNodeID*(), is not the same as its intended destination node. *(b)* A DEP-quadruple by the agent itself at its current step already exists on the node. *(c)* The ARR-quadruple that the agent wishes to insert into *WB.list* is already there. *(d)* One of the agents which reported their departure from the previous node has not reported its arrival at the current node.

Note that, by waiting for agents already present at the node to conclude their interaction with the whiteboard before initiating its own, the algorithm guarantees that an agent which is killed by the malicious host while holding the whiteboard lock will also cause future agents visiting the host to effectively kill themselves, as they will keep requesting the whiteboard lock forever. Moreover, the conditions *(a)*–*(c)* ensure that if the malicious host forwards an agent to the wrong node, or does not forward an agent at all and pretends to be the destination node, or attempts to re-forward a duplicate copy of an agent, then the offended agent will detect this and kill itself instead of continuing the protocol erroneously and disrupting the entire system. Finally, condition *(d)* ensures that if the malicious host disrupts the FIFO order of its queues, the agents which are pushed forward in the queues will detect this and kill themselves.

We now give a high-level description of the algorithm. An agent is always in one of two modes: *clockwise* (+) or *counterclockwise* (−). In any configuration of the system in which a node u contains an entry of the form $\langle \mathrm{id}, + \rangle$ (resp. $\langle \mathrm{id}, - \rangle$) in *WB.flags*, we will say that u contains the flag u^+ (resp. u^-), or that the flag u^+ (resp. u^-) is present. An agent in clockwise mode performs consecutive *cautious steps* in the clockwise direction, until it detects a node w with a flag (either w^+ or w^-), at which point it *bounces* and starts performing cautious steps in the counterclockwise direction. Let u be a node and v its clockwise neighbor. A cautious step starting from u in the clockwise direction entails the following sequence of operations:

- An Explore(+) step: The agent inserts a flag $\langle \mathrm{id}, + \rangle$ and moves to v.
- A Return(+) step: The agent inserts a flag $\langle \mathrm{id}, - \rangle$ and moves back to u.
- A Finish(+) step: The agent removes its $\langle \mathrm{id}, + \rangle$ flag, moves to v, removes its $\langle \mathrm{id}, - \rangle$ flag, and then starts an Explore(+) step from v.

However, if after the Explore(+) step the agent detects a flag at v, then it performs a Bounce(+) step instead: The agent moves back to u without inserting a flag in the whiteboard of v, removes its $\langle \mathrm{id}, + \rangle$ flag, and then either switches to counterclockwise mode and starts an *explore* step in the counterclockwise direction if there is no u^- flag, or remains in clockwise mode and starts an *explore* step in the clockwise direction if there is a u^- flag.

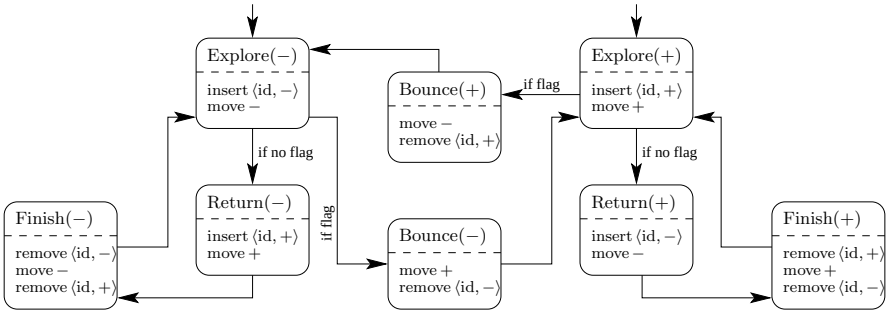

Fig. 2. A diagram of the basic operations of Algorithm PDR_Rings_gray$^+$. "move +" (resp. "move −") stands for traversing an edge in the clockwise (resp. counterclockwise) direction.

An agent in counterclockwise mode operates in a completely symmetric fashion and performs consecutive cautious steps in the counterclockwise direction, until it bounces and switches to clockwise mode. Note that an agent can start the algorithm in clockwise or counterclockwise mode: this is decided when the agent begins its execution, depending on which flags are present on the homebase. Figure 2 illustrates the high-level workings of the algorithm.

The next lemma is a straightforward consequence of our algorithm.

Lemma 9. *Under any execution of* PDR_Rings_gray$^+$ *in which not all agents are killed, Periodic Data Retrieval is achieved.*

Proof. According to the algorithm, an agent never permanently leaves a flag at a node, unless this agent is killed. Also an agent located at a node u either leaves a flag at u (before starting an Explore step) or leaves a flag at a node v, which is adjacent to u (before starting a Return step). Then this agent must pick up those flags before it can leave new flags. Therefore a flag can be permanently left only at the malicious node or at its neighbours. Hence the nodes adjacent to the malicious node may have at most one permanent flag each (left by an agent traversing the edge incident to the malicious node). Also no agent waits infinitely at a safe node. Notice also that an agent moves freely in any area of consecutive nodes as long as those nodes do not contain flags placed by other agents. Furthermore, consider an agent A visiting a node with a flag temporarily placed by another agent B; then this node will eventually be again visited by B. Hence any node u apart from the malicious node and its neighbours is visited infinitely many times clockwise and counter-clockwise by an agent (maybe even the same agent). The nodes adjacent to the malicious node are also visited infinitely many times by agents coming from the safe area. Finally, an agent cannot be permanently trapped in the area which consists of the malicious node and its neighbours and staying alive, since if such an agent traverses one of the edges incident to the malicious node then it will either escape or kill itself and

if an agent A is located at the malicious node and cannot start an Explore step because the malicious node has two permanent flags, then A kills itself. □

In order to show that PDR_Rings_gray$^+$ works with four agents, we reason as follows: First, we show that under any benign execution, at most three agents can be in the queues of the same node at the same time (Lemma 13 below). For this, we take advantage of the flags left by the agents during the cautious step. Then, we show how to convert any execution in which four agents die into a benign execution in which all four agents are in the queues of the malicious host at the same time. This contradicts the immediately preceding statement, thus the malicious host cannot kill four agents, and thus, by Lemma 9, the $(4, gray^+)$-correctness of the algorithm follows (Theorem 4 below). The low-level operations of the algorithm play a crucial role in the proof of Theorem 4.

An agent A arriving at a node u (following the algorithm) executes the following steps:

- Agent A is removed from an incoming-port queue of u and it is inserted in the whiteboard queue of u (atomic step),
- then at some point, agent A is at the front of the whiteboard queue of u, is granted access to the whiteboard of u and writes the ARR-quadruple on u (atomic step),
- then agent A writes the DEP-quadruple on u, it is removed from the whiteboard queue of u and it is inserted in an outgoing-port queue of u (atomic step),
- then agent A is inserted in an incoming-port queue of a node v.

Definition 10. *We say that an agent is* going from u to v *if it has written its* DEP*-quadruple on u and requested to be transferred to v, but has not yet written its corresponding* ARR*-quadruple on v. This means that the agent could be either in the outgoing queue of u, or in the process of being transferred to v, or in the incoming queue of v, or in the whiteboard queue of v. An agent is* traversing an edge (u, v) *if it is going from u to v or from v to u. An agent is* on a node u *if it has written its* ARR*-quadruple on u but has not yet written its* DEP*-quadruple.*

Proposition 9 below states an easy to check property of the algorithm.

Proposition 9. *Under any benign execution of* PDR_Rings_gray$^+$*, at most one agent can be on a given node at a given time. This is the agent at the front of the whiteboard queue of the given node.*

Let A be an agent which is making a move from node u to a neighboring node v, i.e., A has inserted a DEP-quadruple at u but has not yet inserted the corresponding ARR-quadruple at v. If this move is part of an Explore(+) step, we assign to agent A the tag E^+. Similarly, we use the tags R^+, F^+, and B^+ for the Return(+), Finish(+), and Bounce(+) steps, respectively, and the tags E^-, R^-, F^-, and B^- for the symmetric counterclockwise-mode steps.

By a careful case analysis, we can show that, if two agents are traversing the same edge in any direction, then the only possible combinations of tags are:

$\{E^+, B^-\}$, $\{E^+, F^+\}$, $\{E^-, B^+\}$, $\{E^-, F^-\}$, $\{E^+, E^-\}$, and $\{B^+, B^-\}$ (Lemmas 10 and 11 below). Using this characterization, we can prove Lemma 12.

Lemma 10. *Under any benign execution of* PDR_Rings_gray$^+$*, if two agents are traversing the same edge in the same direction, the only possible combinations of tags are the following:* $\{E^+, B^-\}$, $\{E^+, F^+\}$, $\{E^-, B^+\}$, *and* $\{E^-, F^-\}$.

Proof. We eliminate the impossible tag combinations by a case analysis. Let u, v be two neighboring nodes in clockwise order.

- The combination $\{E^+, E^+\}$ can never appear on (u, v), because no agent initiates an Explore(+) step on a node that already contains a "+" flag. This follows easily from the algorithm description.
- Suppose that the combination $\{E^+, R^-\}$ appears on (u, v). This implies that u contains one u^+ flag from each agent. However, the two flags cannot have been left at the same time. If the E^+ agent was the first to leave its u^+ flag, then the other agent, upon seeing that flag on u, would perform a Bounce(−) step instead of a Return(−) step. Similarly, if the R^- agent was the first to leave its u^+ flag, then the other agent would perform a Bounce(+) step instead of an Explore(+) step.
- Suppose that the combination $\{F^+, F^+\}$ appears on (u, v). It follows that there are two v^- flags, left by two earlier Return(+) steps from the two agents. However, the second agent to perform its Return(+) step would have seen the first v^- flag and have performed a Bounce(+) step instead.
- Suppose that the combination $\{F^+, R^-\}$ appears on (u, v). This implies that v contains one v^- flag from each agent, one left during an earlier Return(+) step from the F^+ agent, and the other left during an earlier Explore(−) step from the R^- agent. However, the two flags cannot have been left at the same time. If the F^+ agent was the first to leave its v^- flag, then the other agent, upon seeing that flag on v, would never perform an Explore(−) step from v. Similarly, if the R^- agent was the first to leave its v^- flag, then the other agent would never perform a Return(+) step from v.
- The combination $\{F^+, B^-\}$ is impossible for the same reason as the previous combination. The B^- agent must have executed an earlier Explore(−) step, which conflicts with the F^+ agent's earlier Return(+) step.
- The combinations $\{R^-, R^-\}$, $\{R^-, B^-\}$, and $\{B^-, B^-\}$ are also impossible, because in all cases both agents must have executed an earlier Explore(−) step. However, the second agent to initiate that step would never initiate it, in view of the v^- flag left by the first agent.
- The remaining combinations $\{E^-, E^-\}$, $\{E^-, R^+\}$, $\{F^-, F^-\}$, $\{F^-, R^+\}$, $\{F^-, B^+\}$, $\{R^+, R^+\}$, $\{R^+, B^+\}$, and $\{B^+, B^+\}$ are symmetric to the above and can be eliminated using symmetric arguments.

□

Lemma 11. *Under any benign execution of* PDR_Rings_gray$^+$*, if two agents are traversing the same edge in different directions, the only possible combinations of tags are the following:* $\{E^+, E^-\}$ *and* $\{B^+, B^-\}$.

Proof. We eliminate the impossible tag combinations by a case analysis. Let u, v be two neighboring nodes in clockwise order.

- Suppose that one of the combinations $\{E^+, R^+\}$, $\{E^+, B^+\}$, or $\{E^+, F^-\}$ appears on (u, v). In all cases, the agent moving from v to u must have performed an Explore(+) step from u or a Return(−) step at an earlier time, leaving a u^+ flag. The agent moving from u to v performing an Explore(+) step, also leaves a u^+ flag at u. However, the second of the two agents to leave a u^+ flag at u would have seen the first agent's u^+ flag and would not have performed that step.
- Suppose that one of the combinations $\{F^+, R^+\}$, $\{F^+, B^+\}$ appears on (u, v). In both cases, the agent moving from v to u must have performed an Explore(+) step from u at an earlier time, which must have happened after the F^+ agent started its Finish(+) step and removed its u^+ flag. However, because of the FIFO links, this means that the R^+/B^+ agent is still performing its Explore(+) step, which is a contradiction.
- Suppose that the combination $\{F^+, F^-\}$ appears on (u, v). Earlier, the F^+ agent must have performed an Explore(+) step from u, during which it arrived at v and checked for the existence of a flag. If this check was performed before the F^- agent started its Explore(−) step, then the F^+ agent left a v^- flag, which would prevent the F^- agent from ever initiating its Explore(−) step. On the other hand, if the check was performed after the F^- agent started its Explore(−) step, then in fact the check must have happened after the F^- started its Finish(−) step and removed its v^- flag (otherwise the F^+ agent would have bounced). In this case, the F^+ agent cannot have concluded its Return(+) step because of the FIFO property of the links.
- Suppose that one of the combinations $\{R^-, R^+\}$, $\{R^-, B^+\}$ appears on (u, v). In both cases, both agents have left a u^+ flag at an earlier time. If the R^- agent left its u^+ flag first, then the other agent would never have initiated an Explore(+) step. On the other hand, if the R^+/B^+ agent left its u^+ flag first, then the R^- agent would have bounced instead of performing a Return(−) step.
- The remaining combinations $\{E^-, R^-\}$, $\{E^-, B^-\}$, $\{E^-, F^+\}$, $\{F^-, R^-\}$, $\{F^-, B^-\}$, $\{R^+, B^-\}$ are symmetric to the above and can be eliminated using symmetric arguments.

□

Lemma 12. *Under any benign execution of* PDR_Rings_gray$^+$*, it is not possible for three agents to traverse the same edge at the same time.*

Proof. Let $G = (V, E)$, with vertices $V = \{B^-, B^+, E^-, E^+, F^-, F^+\}$ and edges $E = \{\{E^+, F^+\}, \{E^-, B^+\}, \{E^-, F^-\}, \{E^+, B^-\}, \{E^+, E^-\}, \{B^+, B^-\}\}$ (Figure 3). By Lemmas 10 and 11, whenever two agents traverse the same edge of the ring C_n at the same time, their tags must necessarily be connected by an edge in G. It follows that, if three agents traverse the same edge at the same time, then their tags must induce a triangle in G. The lemma follows by the observation that G does not contain any triangles. □

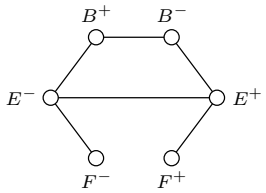

Fig. 3. The graph G in the proof of Lemma 12.

Lemma 12 and Proposition 9 considerably limit the candidate configurations of four agents in the queues of the same node. By a more elaborate case analysis, we can also eliminate the remaining possibilities and arrive at a contradiction in all cases, thus obtaining the following:

Lemma 13. *For any node v other than the homebase, under any benign execution of* PDR_Rings_gray$^+$*, a total of at most three agents can be in the queues of v at the same time.*

Proof. Let u, v, w be consecutive nodes in clockwise order. Suppose that, under some benign execution, four agents are in the queues of v at the same time. By Lemma 12 and Proposition 9, the only candidate configurations are the following: *(a)* Two agents are traversing the edge (u, v) and two agents are traversing the edge (v, w), *(b)* two agents are traversing the edge (u, v), one agent is traversing the edge (v, w), and one agent is on v, and *(c)* two agents are traversing the edge (v, w), one agent is traversing the edge (u, v), and one agent is on v. Since *(b)* and *(c)* are symmetric, it suffices to derive a contradiction for *(a)* and *(b)*.

We first remark that it follows from Lemmas 10 and 11 that whenever two agents are traversing the same edge, there exist flags on both endpoints of that edge. We now distinguish the two cases:

Configuration (a). Since we have one pair of agents traversing each of the edges (u, v) and (v, w), it follows that node v contains both flags v^+ and v^-. Without loss of generality, we assume that v^+ was left first. By Lemmas 10 and 11, the v^- flag was left after the v^+ flag either by some agent executing an Explore($-$) step from v, or by some agent executing a Return($+$) step from v. However, since v is not the homebase, no agent can start an Explore($-$) step from v when there is already a v^+ flag on v (this is an easy property of the algorithm). Moreover, any agent would perform a Bounce($+$) step instead of a Return($+$) step upon perceiving the flag v^+. Therefore, we have a contradiction.

Configuration (b). Let A, B be the two agents traversing the edge (u, v). Of those, let A be the one that left the u^+ flag and let B be the one that left the v^- flag. Let C be the agent on node v and D be the agent traversing the edge (v, w). Since v is not the homebase, C must have arrived at v either from u or from w.

– If C arrived from u, then it must have performed one of the following steps from u: Explore($+$), Finish($+$), Return($-$), or Bounce($-$). Explore($+$) and

Return(−) can be eliminated immediately, since they imply that C has left a u^+ flag which conflicts with the u^+ flag of agent A. Bounce(−) is also eliminated because it implies an earlier Explore(−) step from v which resulted in C leaving a v^- flag, and this conflicts with the v^- flag left by B. Finally, Finish(+) is eliminated because it implies that C left a v^- flag in an earlier Return(+) step from v, and this conflicts with the v^- flag left by B.

- If C arrived from w, then we can rule out the possibility of arriving by performing a Bounce(+), Return(+), or Finish(−) step, since in all cases it would mean that C has earlier left a v^+ flag, which conflicts with the v^- flag left by B. It follows that C must have arrived at v by an Explore(−) step, and therefore there exists a w^- flag on w. But this leaves no option for agent D: whatever step it may be performing, it must have left a flag on v or w, conflicting with the flags left there by B and C.

We thus have a contradiction in all cases. □

Theorem 4. PDR_Rings_gray$^+$ *is* $(4, gray^+)$*-correct in rings.*

Proof. Let ω be the malicious host. In view of Lemma 9, it suffices to prove that ω cannot kill all four agents. Suppose that there exists an execution $\mathcal{E}$ under which four agents are killed. We will show that there exists an execution $\mathcal{E}'$ such that at any time all whiteboard contents are the same as under $\mathcal{E}$, and furthermore the agents are not killed but frozen on ω or in its queues, up to the time when the last agent arrives there. Since $\mathcal{E}'$ is a benign execution under which four agents are in the queues of ω at the same time, we get a contradiction by Lemma 13.

We now show how to construct $\mathcal{E}'$. Assume that the first agent A is killed at time t_A under $\mathcal{E}$. We distinguish cases according to how A is killed.

- If A is killed while it is on ω (i.e., having written its ARR-quadruple but not the corresponding DEP-quadruple), then in $\mathcal{E}'$ we freeze A at time t_A. Note that this does not make any difference in the whiteboards with respect to the execution $\mathcal{E}$. Moreover, under $\mathcal{E}$, any agent that visits ω after t_A is stuck waiting for A to write its DEP-quadruple. The same happens under $\mathcal{E}'$.
- If A is killed while it is in one of the incoming queues of ω, under $\mathcal{E}'$ we freeze A while it is in the incoming queue, effectively ignoring that queue after time t_A. Again, note that this does not make any difference in the whiteboards with respect to $\mathcal{E}$. Moreover, under $\mathcal{E}$, any subsequent incoming agent from the same port would realize that A has not written yet its ARR-quadruple on ω and kill itself. Effectively the same thing happens under $\mathcal{E}'$, since any subsequent incoming agent from the same port is stuck in the incoming queue after A.
- If A is killed in an outgoing queue of ω, under $\mathcal{E}'$ we freeze it in the outgoing queue, effectively ignoring that queue after time t_A. By the same reasoning as in the previous case, we can continue $\mathcal{E}'$ by replicating $\mathcal{E}$, except that A is now frozen instead of killed.

- If A kills itself either on ω or on one of its neighbors because it detects that it was erroneously transferred, then under $\mathcal{E}'$ we continue by putting it in the correct outgoing queue of ω and then freezing it and ignoring the queue. Again, this does not change the whiteboards with respect to $\mathcal{E}$ and subsequent agents using that outgoing queue will behave in the same manner under both $\mathcal{E}$ and $\mathcal{E}'$.
- If A kills itself because it detects a violation of the FIFO order of a queue, then under $\mathcal{E}'$ we execute the agents that were before A in the queue as they were executed under $\mathcal{E}$, and when A is at the front of the queue we freeze it and the queue.

If ω attempts to forward a duplicate copy of an agent under $\mathcal{E}$, then under $\mathcal{E}'$ we simply ignore that action. This is safe to do, since the duplicate agent has no effect on the whiteboards and kills itself immediately under $\mathcal{E}$.

We now have an execution which is benign up to the point when the second agent dies. By performing the above modifications again for the second and the third agent to die, we get a benign execution in which three agents are in the queues of ω and the fourth agent necessarily visits it at some point, for it is only by visiting ω that an agent can be killed. When the fourth agent visits ω, we have a benign execution $\mathcal{E}'$ under which four agents are in the queues of ω at the same time. □

5 An Efficient Algorithm for Rings with a Red Hole

When the malicious host is a red hole, the PDR_Rings_gray$^+$ algorithm, that we presented in Section 4, fails irrespective of the number of agents. Indeed, the red hole can kill every clockwise (resp. counterclockwise) agent that approaches it after it has removed the $+$ (resp. $-$) flag from the neighboring node and while it is concluding its Finish$(+)$ (resp. Finish$(-)$) step on the red hole, by presenting to it a whiteboard which shows that previous clockwise (resp. counterclockwise) agents were not killed but continued their intended trajectory.

In order to remedy this situation, we propose algorithm PDR_Rings_Red, which employs a natural extension of the cautious step idea: the *cautious double step*. Let u, v, w be consecutive nodes in clockwise order. A cautious double step starting from u in the clockwise direction entails the following sequence of operations:

- An Explore1$(+)$ step: The agent inserts a flag $\langle \text{id}, + \rangle$ and moves to v.
- An Explore2$(+)$ step: The agent moves to w.
- A Return2$(+)$ step: The agent moves back to v.
- A Return1$(+)$ step: The agent moves back to u.
- A Finish$(+)$ step: The agent removes its $\langle \text{id}, + \rangle$ flag, moves to v, and then starts an Explore1$(+)$ step from v.

However, if after the Explore1$(+)$ step the agent detects a $+$ flag at v, then it bounces but first it goes to w anyway. More specifically, in this case the agent performs the following sequence of operations after the Explore1$(+)$ step:

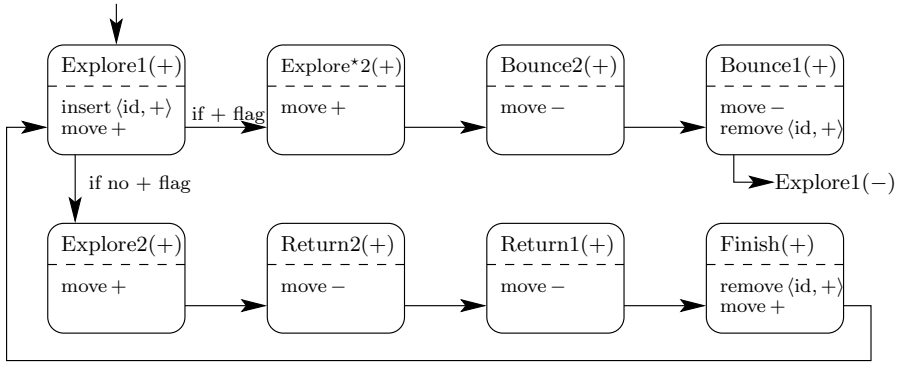

Fig. 4. A diagram of the basic operations of Algorithm PDR_Rings_Red. "move +" (resp. "move −") stands for traversing an edge in the clockwise (resp. counterclockwise) direction.

- An Explore*2(+) step: The agent moves to w.
- A Bounce2(+) step: The agent moves back to v.
- A Bounce1(+) step: The agent moves back to u, removes its $\langle \mathrm{id}, + \rangle$ flag, and then starts an Explore1(−) step from u.

Under PDR_Rings_Red, a clockwise agent performs consecutive cautious double steps in the clockwise direction, until it bounces and switches to counterclockwise mode. We notice here that an agent will not start an Explore1(+) or an Explore1(−) step at a node u if there is a u^+ or u^- flag there respectively. A counterclockwise agent operates completely symmetrically. Figure 4 illustrates the high-level workings of the algorithm.

We should mention at this point that the low-level operations performed by this algorithm when an agent moves from node to node are somewhat more contrived than in the previous case. We highlight the differences below:

- The ARR- and DEP-tuples now contain more information, namely the mode of the agent (clockwise or counterclockwise) and the name of the step which it is currently executing (one of Explore1, Explore2, Return2, Return1, Finish, Explore*2, Bounce2, Bounce1).
- The agent keeps copies of *WB.list* from each node before every step, and after every step checks its stored copies against the whiteboard of the current node for inconsistencies. This allows the agent to verify that each whiteboard is consistent with the whiteboards of its neighbors, as well as that it reports a correct execution of the protocol. If any inconsistency is detected, the agent halts (kills itself). This check supersedes the simpler check in PDR_Rings_gray$^+$, whereby the agent simply checked whether all agents previously departed from the same port had reached the destination.

A feature of the algorithm is that clockwise agents ignore the flags of counterclockwise agents (i.e., they do not bounce upon detecting such a flag) and vice

versa. This leads to an algorithm which is likely suboptimal, but can be analyzed more easily by considering the deaths of clockwise agents separately from those of counterclockwise agents. In fact, one can show that under any execution, we can have at most three deaths of clockwise agents and, symmetrically, at most three deaths of counterclockwise agents.

Lemma 14. *Under any execution of* PDR_Rings_Red *in a ring with at least* 6 *nodes, at most 6 agents die.*

Proof. Let u, v, ω, x, y be consecutive nodes in clockwise order, where ω is the red hole. For the purposes of this proof, call an agent that is killed by ω *clockwise* (resp. *counterclockwise*) if its last Explore1 step was an Explore1(+) step initiated from u or v (resp. Explore1(−) step initiated from x or y).

We first argue that at most one clockwise agent may be killed by ω without leaving any flag on u or v. Note that an agent won't leave any flag on u or v if it dies either while completing a Finish(+) step of its cautious double step from v, or during its cautious double step from ω itself. Suppose that a clockwise agent A died leaving no flags on u or v and consider the next clockwise agent B to be killed by ω. If B died during a cautious double step from u after A's death, it is clear that a u^+ flag will remain. Assuming that B started an Explore1(+) step from v after A's Finish(+) step, we now argue that it is not possible for B to remove its v^+ flag: Note that, once B realizes that the protocol has not been executed correctly, it dies leaving its flag. Therefore, ω has three options: (a) to kill B outright, (b) to present to B a whiteboard that says that A hasn't removed its ω^+ flag yet, or (c) to present an appropriate whiteboard to B, that says that A concluded its Finish(+) step, removed its ω^+ flag, and moved on to x. Under (a), B will clearly not remove its v^+ flag. Under (b), B will decide to bounce and therefore the only chance for ω to kill it is before it returns to v, so its v^+ flag will remain. Under (c), agent B will subsequently visit x and realize that the protocol has not been executed correctly. So, in all cases, the v^+ flag left by B remains. Moreover, B's Explore1(+) step cannot have occurred earlier than A's, because then A would either bounce or realize something is wrong upon reaching x or returning to v.

Next, observe first that if any clockwise agent dies leaving a u^+ flag, then all subsequent clockwise agents will detect the flag at u, go up to v, and then bounce back without being killed as they do not visit ω. Also, if any clockwise agent leaves a v^+ flag and dies, then at most one more agent may be killed by ω. This is because all subsequent clockwise agents will start a cautious double step from u, detect the v^+ flag, visit ω, and then bounce. Note that whenever the red hole decides to kill one of these subsequent agents its u^+ flag will remain permanently on u, thus preventing any other clockwise agent from dying.

The proof is complete because by the above arguments the red-hole can kill at most one clockwise agent leaving no flags, at most one leaving a flag at v, and at most one leaving a flag at u, and similarly for counterclockwise agents. □

We note here that 6 agents can indeed be killed by executing Algorithm PDR_Rings_Red as follows. Let u, v, ω, x, y be consecutive nodes in clockwise

order, where ω is the red hole. One clockwise agent and one counterclockwise agent start their cautious double steps from v and x respectively at the same time and both are killed before or after completing their Finish steps. Then only agents starting their cautious double steps at v (clockwise) or x (counterclockwise) can detect the dead agents. Hence those agents can be killed leaving a flag v^+ and a flag x^-. Then another two agents which start their cautious double steps at u, y can be killed.

As in the case of PDR_Rings_gray$^+$, the agent never walks too far away from its flag. The agent is always at distance at most 2 from a flag that it has left behind. In fact, if the agent does not return to pick up the flag, then this must have happened because it was killed as a result of malicious activity from the red hole. Therefore, any flag which remains forever on a node after a given point in time is at distance at most 2 from the red hole. This, together with the fact that even when an agent decides to bounce, it still goes one step further in its intended direction (step Explore*2), implies the following:

Lemma 15. *Under any execution of* PDR_Rings_Red *with at least* 7 *agents in a ring of at least* 6 *nodes, Periodic Data Retrieval is achieved.*

Proof. Suppose that $k \geq 7$ agents execute the Algorithm PDR_Rings_Red. In view of Lemma 14 at least $k-6$ agents remain alive. We will prove that $k-6$ agents are always acting in the safe area of consecutive nodes which lies between the two neighbours of the malicious node (i.e., the area not passing from the malicious node). We will also prove that none of those $k-6$ agents is permanently blocked at a distance two from the malicious node.

Let u, v, ω, x, y be consecutive nodes in clockwise order, where ω is the red hole. According to the algorithm, an agent never permanently leaves a flag at a node, unless this agent is killed. Also an agent located at a node leaves a flag at this node before starting an Explore step. Then this agent must pick up this flag before it can leave a new flag. Therefore flags can be permanently left only at the nodes u, v, ω, x, y. Moreover, in a ring of at least 6 nodes, u, v can only have u^+ and v^+ permanent flags and x, y can only have x^- and y^- permanent flags. Also no agent waits infinitely at a safe node. Notice also that an agent moves freely in any area of consecutive nodes as long as those nodes do not contain flags placed by other agents. Furthermore, consider an agent A visiting a node with a flag temporarily placed by another agent B; then this node will eventually be again visited by B. Hence any node apart from nodes u, v, ω, x, y is visited infinitely many times clockwise and counter-clockwise by an agent (maybe even the same agent).

In view of Lemma 14, no more than six agents vanish in the red hole and therefore after that there are at least $k-6 \geq 1$ agents acting in the safe area between nodes u, y and there are permanent flags u^+ and y^- at u and y respectively left by agents which vanished. To see this, first notice that if there is an agent A acting in the safe area between nodes u, y and either u does not have a flag u^+ or y does not have a flag y^- then agent A can start either an Explore1$(+)$ or an Explore1$(-)$ step from u or y respectively after which will end

up in the red hole. But in that case more than six agents can be killed, violating Lemma 14. All those $k-6$ agents act in the safe area between nodes u, y since the agents cannot be blocked anywhere else than node ω (remember that nodes u, v can only have u^+ and v^+ permanent flags and nodes x, y can only have x^- and y^- permanent flags) and in that case again more than six agents would be killed.

Nodes u, v, x, y are also visited infinitely many times by agents coming from the safe area. □

Observe that for rings with $n < 6$ nodes, the problem can always be solved by $n-1$ agents: each agent selects a different node (except the home base) and infinitely visits all other nodes. By this observation, and Lemmas 14 and 15 we obtain that PDR_Rings_Red achieves Periodic Data Retrieval with seven agents:

Theorem 5. PDR_Rings_Red *is* $(7, red)$*-correct in rings.*

Remark 6. Note that the red hole might not interfere with the agents in any way, except by modifying the data items that they store in its whiteboard. In this case, it could happen that altered or corrupted data from certain nodes reach the homebase, thus rendering the algorithm incorrect. However, the cautious double step ensures that any agent which leaves a data item on the red hole will also have the option to leave a copy of it on at least one of its neighbours. Therefore, by enforcing agents to disregard a data item which they find in two different versions on two neighbouring nodes, we ensure that an agent will never deliver a corrupted data item from the whiteboard of the red hole to any other node. In other words, an agent at a node u:

- Picks up data produced by u and copies those data at any node it visits (until it visits the homebase). If the agent was already carrying data produced by node u, then it disregards the previously carried data (obviously, if the two data sets for the same period are different, then the previously carried data must have been found at the red hole). Note that data produced by the red hole itself will also be picked up and copied. But those are not corrupted data which had been produced elsewhere.
- Picks up data which were previously copied in u but had been produced in any other node $v \neq u$ (if any). It compares them with data found in neighbouring nodes it visits until either it finds exactly the same data, or it finds a different dataset produced by v for the same period. In the former case it copies data at any node it visits (until it visits the homebase). In the latter case it disregards both datasets.

6 Concluding Remarks

We gave the first nontrivial lower bounds on the number of agents for Periodic Data Retrieval in asynchronous rings with either one gray hole or one red hole, answering an open question posed in [20]. Moreover, we proposed an optimal,

with respect to the number of agents, protocol for Periodic Data Retrieval in asynchronous rings with a gray hole, improving the previous upper bound of 9 agents and settling the question of the optimal number of agents in the gray-hole case. Finally, we proposed a protocol working with 7 agents in the presence of a red hole, significantly improving the previously known upper bound of 27 agents.

We made no effort to optimize the amount of data stored on the whiteboards of the hosts. Indeed, since the protocol is executed indefinitely, the amount of data stored in every host under both PDR_Rings_gray$^+$ and PDR_Rings_Red grows unbounded. Hence an open question is whether there is a protocol in which the amount of data stored in every host is bounded.

Algorithm PDR_Rings_Red is almost certainly suboptimal. In principle, we should be able to further reduce the total number of agents killed by suitably marking all of the nodes involved in a cautious double step, and then having clockwise and counterclockwise agents *not* ignore each other's flags. We conjecture that an algorithm along these lines would work with an optimal number of 5 agents in the presence of a red hole.

One important research direction which remains completely open is the case of a malicious host which can alter the state of an agent, its memory, or even its program. It would be particularly interesting to develop algorithms that cope with this kind of malicious behaviour. Another question that remains open is what happens in other network topologies under the various malicious host models.

Acknowledgment

We are grateful to two anonymous reviewers for reading the manuscript very carefully and proposing some nice improvements to the presentation of the paper.

References

1. Balamohan, B., Dobrev, S., Flocchini, P., Santoro, N.: Asynchronous exploration of an unknown anonymous dangerous graph with O(1) pebbles. In: Even, G., Halldórsson, M.M. (eds.) SIROCCO. Lecture Notes in Computer Science, vol. 7355, pp. 279–290. Springer (2012)
2. Balamohan, B., Flocchini, P., Miri, A., Santoro, N.: Time optimal algorithms for black hole search in rings. Discrete Math., Alg. and Appl. 3(4), 457–472 (2011)
3. Chalopin, J., Das, S., Santoro, N.: Rendezvous of mobile agents in unknown graphs with faulty links. In: Pelc, A. (ed.) DISC. Lecture Notes in Computer Science, vol. 4731, pp. 108–122. Springer (2007)
4. Czyzowicz, J., Dobrev, S., Gąsieniec, L., Ilcinkas, D., Jansson, J., Klasing, R., Lignos, I., Martin, R., Sadakane, K., Sung, W.K.: More efficient periodic traversal in anonymous undirected graphs. Theor. Comput. Sci. 444, 60–76 (2012)
5. Dobrev, S., Flocchini, P., Královič, R., Ruzicka, P., Prencipe, G., Santoro, N.: Black hole search in common interconnection networks. Networks 47(2), 61–71 (2006)

6. Dobrev, S., Flocchini, P., Královič, R., Santoro, N.: Exploring an unknown graph to locate a black hole using tokens. In: Navarro, G., Bertossi, L.E., Kohayakawa, Y. (eds.) IFIP TCS. IFIP, vol. 209, pp. 131–150. Springer (2006)
7. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Mobile search for a black hole in an anonymous ring. In: Welch, J.L. (ed.) DISC. Lecture Notes in Computer Science, vol. 2180, pp. 166–179. Springer (2001)
8. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Multiple agents rendezvous in a ring in spite of a black hole. In: Papatriantafilou, M., Hunel, P. (eds.) OPODIS. Lecture Notes in Computer Science, vol. 3144, pp. 34–46. Springer (2003)
9. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Searching for a black hole in arbitrary networks: optimal mobile agents protocols. Distributed Computing 19(1), 1–19 (2006)
10. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Mobile search for a black hole in an anonymous ring. Algorithmica 48(1), 67–90 (2007)
11. Dobrev, S., Královič, R., Santoro, N., Shi, W.: Black hole search in asynchronous rings using tokens. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC. Lecture Notes in Computer Science, vol. 3998, pp. 139–150. Springer (2006)
12. Dobrev, S., Santoro, N., Shi, W.: Scattered black hole search in an oriented ring using tokens. In: IPDPS. pp. 1–8. IEEE (2007)
13. Dobrev, S., Santoro, N., Shi, W.: Using scattered mobile agents to locate a black hole in an un-oriented ring with tokens. Int. J. Found. Comput. Sci. 19(6), 1355–1372 (2008)
14. Flocchini, P., Ilcinkas, D., Santoro, N.: Ping pong in dangerous graphs: Optimal black hole search with pebbles. Algorithmica 62(3-4), 1006–1033 (2012)
15. Flocchini, P., Kellett, M., Mason, P.C., Santoro, N.: Map construction and exploration by mobile agents scattered in a dangerous network. In: IPDPS. pp. 1–10. IEEE (2009)
16. Flocchini, P., Kellett, M., Mason, P.C., Santoro, N.: Searching for black holes in subways. Theory Comput. Syst. 50(1), 158–184 (2012)
17. Flocchini, P., Mans, B., Santoro, N.: Sense of direction: Definitions, properties, and classes. Networks 32(3), 165–180 (1998)
18. Flocchini, P., Mans, B., Santoro, N.: Sense of direction in distributed computing. Theor. Comput. Sci. 291(1), 29–53 (2003)
19. Flocchini, P., Santoro, N.: Distributed security algorithms for mobile agents. In: Cao, J., Das, S.K. (eds.) Mobile Agents in Networking and Distributed Computing, chap. 3, pp. 41–70. John Wiley & Sons, Inc., Hoboken, NJ, USA (2012)
20. Královič, R., Miklík, S.: Periodic data retrieval problem in rings containing a malicious host. In: Patt-Shamir, B., Ekim, T. (eds.) SIROCCO. Lecture Notes in Computer Science, vol. 6058, pp. 157–167. Springer (2010)
21. Markou, E.: Identifying hostile nodes in networks using mobile agents. Bulletin of the EATCS 108, 93–129 (2012)
22. Miklík, S.: Exploration in faulty networks. Ph.D. thesis, Comenius University, Bratislava (2010)
23. Shi, W.: Black hole search with tokens in interconnected networks. In: Guerraoui, R., Petit, F. (eds.) SSS. Lecture Notes in Computer Science, vol. 5873, pp. 670–682. Springer (2009)

A Algorithm PDR_Rings_gray$^+$

```
    ▷ Agent variables
 1: id                                   ▷ Constant agent identifier set at agent creation
 2: prev_node, dest_node, curr_node      ▷ Node labels
 3: in_port, out_port                    ▷ Port labels
 4: step                                 ▷ Stores number of edge traversals performed
 5: initial_mode                         ▷ Initial agent mode, takes one of the values {+, −}
 6: prev_list                            ▷ Departed agents' entries from previous node
 7: algorithm PDR_Rings_gray+
 8:     WB.access()
 9:     curr_node ← getNodeID()
10:     step ← 0
11:     in_port ← #                     ▷ undefined
12:     initial_mode ← DecideDirection()
13:     Insert ⟨id, ARR, in_port, step⟩ into WB.list
14:     Explore(initial_mode)
15: end algorithm

16: procedure Explore(mode)
17:     Insert ⟨id, mode⟩ into WB.flags
18:     Move(mode)
19:     if WB.flags is not empty then
20:         Bounce(mode)
21:     else
22:         Return(mode)
23:     end if
24: end procedure

25: procedure Return(mode)
26:     Insert ⟨id, Opposite(mode)⟩ into WB.flags
27:     Move(Opposite(mode))
28:     Finish(mode)
29: end procedure

30: procedure Finish(mode)
31:     Remove this agent's tuple from WB.flags
32:     Move(mode)
33:     Remove this agent's tuple from WB.flags
34:     if ∃k s.t. ⟨k, mode⟩ ∈ WB.flags then   ▷ Can happen only if current node is the homebase
35:         Explore(Opposite(mode))
36:     else
37:         Explore(mode)
38:     end if
39: end procedure
```

```
40: procedure Bounce(mode)
41:     Move(Opposite(mode))
42:     Remove this agent's tuple from WB.flags
43:     if ∃k s.t. ⟨k, Opposite(mode)⟩ ∈ WB.flags then
                                        ▷ Can happen only if current node is
                                          the homebase
44:         Explore(mode)
45:     else
46:         Explore(Opposite(mode))
47:     end if
48: end procedure

49: procedure Move(dir)
    ▷ Implements all the low-level operations and checks while moving from node to
    node. After this procedure returns, the agent has access to the whiteboard of the
    destination node and has written its ARR-quadruple.
50:     dest_node ← neighbor of curr_node in direction dir
51:     out_port ← port out of curr_node in direction dir
52:     prev_node ← curr_node
53:     prev_list ← WB.list
54:     Insert ⟨id, DEP, out_port, step⟩ into WB.list
55:     transfer(out_port)
56:     WB.access()
57:     curr_node ← getNodeID()
58:     in_port ← port leading to direction Opposite(dir)
59:     if CheckHalt() then
60:         halt                           ▷ Malicious behavior detected
61:     end if
62:     while ∃k, p, p′, s s.t. ⟨k, ARR, p, s⟩ ∈ WB.list and ⟨k, DEP, p′, s⟩ ∉ WB.list do
63:         WB.release()                   ▷ Some agent is in the middle
64:         WB.access()                    ▷ of its computation
65:     end while
66:     Insert ⟨id, ARR, in_port, step + 1⟩ into WB.list
67:     step ← step + 1
68: end procedure

69: function CheckHalt
70:     if curr_node ≠ dest_node then
71:         return true                    ▷ Malicious host forwarded agent to
                                             wrong node
72:     end if
73:     if ⟨id, DEP, out_port, step⟩ ∈ WB.list then
74:         return true                    ▷ Malicious host did not forward agent
                                             and pretends to be destination node
75:     end if
76:     if ⟨id, ARR, in_port, step + 1⟩ ∈ WB.list then
77:         return true                    ▷ Malicious host is attempting to re-
                                             forward a copy of this agent
78:     end if
```

```
79:     if ∃k, s s.t. ⟨k, DEP, out_port, s⟩ ∈ prev_list and ⟨k, ARR, in_port, s + 1⟩ ∉
        WB.list then
80:         return true                          ▷ Some agent attempted to traverse the
                                                   link earlier but has not arrived yet
81:     end if
82:     return false
83: end function

84: function DecideDirection
        ▷ Used at the homebase to decide if the agent will start as a clockwise or a counter-
        clockwise agent. After this function returns, the agent has access to the whiteboard
        of the homebase.
85:     while ∃k, k′ s.t. ⟨k, +⟩ ∈ WB.flags and ⟨k′, −⟩ ∈ WB.flags do
86:         WB.release()
87:         WB.access()
88:     end while
89:     if ∄k s.t. ⟨k, +⟩ ∈ WB.flags then
90:         return +
91:     else
92:         return −
93:     end if
94: end function

95: function Opposite(dir)
96:     if dir = + then
97:         return −
98:     else
99:         return +
100:    end if
101: end function
```

B Algorithm PDR-Rings-Red

▷ *Agent variables*
id ▷ *Constant agent identifier set at agent creation*
prev_node, dest_node, curr_node ▷ *Node labels*
in_port, out_port ▷ *Port labels*
step ▷ *Stores number of edge traversals performed*
initial_mode ▷ *Initial agent mode, takes one of the values* $\{+, -\}$
prev_list0, prev_list1, prev_list2, prev_list3, prev_list4 ▷ *Copy of WB.list before each move of the double step*

algorithm PDR_Rings_Red
  if the ring consists of $n \leq 5$ nodes **then**
    Each of the first $n-1$ awake agents selects a different node of the ring (except the Homebase) and infinitely visits all other nodes
  else
    WB.access()
    curr_node ← *getNodeID*()
    step ← 0
    in_port ← # ▷ *undefined*
    initial_mode ← DecideDirection()
    Insert $\langle$id, ARR, in_port, step, initial_mode, #$\rangle$ into *WB.list*
    Explore1(initial_mode)
  end if
end algorithm

procedure Explore1(mode)
  Insert $\langle$id, mode$\rangle$ into *WB.flags*
  Move(mode, mode, Explore1)
  if $\exists k$ s.t. $\langle k, \text{mode}\rangle \in$ *WB.flags* **then**
    Explore*2(mode)
  else
    Explore2(mode)
  end if
end procedure

procedure Explore2(mode)
  Move(mode, mode, Explore2)
  Return2(mode)
end procedure

procedure Explore*2(mode)
  Move(mode, mode, Explore*2)
  Bounce2(mode)
end procedure

procedure Return2(mode)

```
38:     Move(Opposite(mode), mode, Return2)
39:     Return1(mode)
40: end procedure

41: procedure Return1(mode)
42:     Move(Opposite(mode), mode, Return1)
43:     Finish(mode)
44: end procedure

45: procedure Finish(mode)
46:     Remove this agent's tuple from WB.flags
47:     Move(mode, mode, Finish)
48:     if ∄k s.t. ⟨k, mode⟩ ∈ WB.flags then
49:         Explore1(mode)
50:     else
51:         Explore1(DecideDirection())
52:     end if
53: end procedure

54: procedure Bounce2(mode)
55:     Move(Opposite(mode), mode, Bounce2)
56:     Bounce1(mode)
57: end procedure

58: procedure Bounce1(mode)
59:     Move(Opposite(mode), mode, Bounce1)
60:     Remove this agent's tuple from WB.flags
61:     if ∃k s.t. ⟨k, Opposite(mode)⟩ ∈ WB.flags then
62:         Explore1(mode)
63:     else
64:         Explore1(Opposite(mode))
65:     end if
66: end procedure

67: procedure Move(dir,mode,walk_step)
    ▷ Implements all the low-level operations and checks while moving from node to
    node. After this procedure returns, the agent has access to the whiteboard of the
    destination node and has written its ARR-tuple.
68:     dest_node ← neighbor of curr_node in direction dir
69:     out_port ← port out of curr_node in direction dir
70:     prev_node ← curr_node
71:     if walk_step = Explore1 then
72:         prev_list0 ← WB.list
73:     else if walk_step = Explore2 or walk_step = Explore⋆2 then
74:         prev_list1 ← WB.list
75:     else if walk_step = Return2 or walk_step = Bounce2 then
76:         prev_list2 ← WB.list
77:     else if walk_step = Return1 or walk_step = Bounce1 then
78:         prev_list3 ← WB.list
79:     else if walk_step = Finish then
```

80: prev_list4 ← *WB.list*
81: **end if**
82: Insert ⟨id, DEP, out_port, step, mode, walk_step⟩ into *WB.list*
83: *transfer*(out_port)
84: *WB.access*()
85: curr_node ← *getNodeID*()
86: in_port ← port leading to direction OPPOSITE(dir)
87: **if** CHECKHALT(mode, walk_step) **then**
88: **halt** ▷ *Malicious behavior detected*
89: **end if**
90: **while** $\exists k, p, p', s, m, r$ s.t. $\langle k, \text{ARR}, p, s, m, r\rangle \in$ *WB.list* and $\langle k, \text{DEP}, p', s, m, r\rangle \notin$ *WB.list* **do**
91: *WB.release*() ▷ *Some agent is in the middle*
92: *WB.access*() ▷ *of its computation*
93: **end while**
94: Insert ⟨id, ARR, in_port, step + 1, mode, walk_step⟩ into *WB.list*
95: step ← step + 1
96: **end procedure**

97: **function** CHECKHALT(mode,walk_step)
98: **if** curr_node ≠ dest_node **then**
99: **return true** ▷ *Malicious host forwarded agent to wrong node*
100: **end if**
101: **if** ⟨id, DEP, out_port, step, mode, walk_step⟩ ∈ *WB.list* **then**
102: **return true** ▷ *Malicious host did not forward agent and pretends to be destination node*
103: **end if**
104: **if** ⟨id, ARR, in_port, step + 1, mode, walk_step⟩ ∈ *WB.list* **then**
105: **return true** ▷ *Malicious host is attempting to re-forward a copy of this agent*
106: **end if**
107: **if** the "prev_list"s up to the current walk_step are inconsistent with respect to each other, or with respect to *WB.list* of the current node, or with respect to the correct execution of the protocol **then**
108: **return true**
109: **end if**
110: **return false**
111: **end function**

112: **function** DECIDEDIRECTION
▷ *Used at the homebase or at the end of a Finishstep to decide if the agent will start as a clockwise or a counterclockwise agent. After this function returns, the agent has access to the whiteboard of the node.*
113: **while** $\exists k, k'$ s.t. $\langle k, +\rangle \in$ *WB.flags* and $\langle k', -\rangle \in$ *WB.flags* **do**
114: *WB.release*()
115: *WB.access*()
116: **end while**
117: **if** $\nexists k$ s.t. $\langle k, +\rangle \in$ *WB.flags* **then**
118: **return** +

```
119:     else
120:         return −
121:     end if
122: end function

123: function Opposite(dir)
124:     if dir = + then
125:         return −
126:     else
127:         return +
128:     end if
129: end function
```