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] Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France
evangelos.bampas@labri.fr
[2] Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece
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 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

⋆ Partial support by the ANR project DISPLEXITY (ANR-11-BS02-014).

⋆⋆ 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).

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: 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" to eventually advance 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.*

Due to lack of space, all missing proofs, as well as the detailed pseudocode for the proposed algorithms, are deferred to the full version of the paper.

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 identified by a unique label, and 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 graph G.

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. 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. 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.
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. 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:*

- 1-malicious *or* black hole *if it kills every agent that appears in any of its queues at every time* $t \geq 0$.
- 2-malicious *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.*
- 3-malicious *or* 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* $t \geq 0$.
- 4-malicious *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* $t \geq 0$.
- 5-malicious *or* 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.

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 Periodic Data Retrieval is a tuple* $\langle G, \lambda, H, k, \omega, m \rangle$*, where* G *is an undirected graph,* λ *is a function that assigns labels to nodes and local ports of the nodes,* $H \in V(G)$ *is the homebase,* k *is a positive integer representing the number of agents starting on the homebase,* $\omega \in V(G) \setminus \{H\}$ *is the malicious host, and* $m \in \{1, 2, 3, 4, 5\}$ *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 $\mathcal{I} = \langle G, \lambda, H, k, \omega, m \rangle$, for every execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I}$, for every node $v \in V(G) \setminus \{\omega\}$, 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 $\omega \notin \{v_0, \ldots, v_r\}$.

Propositions 1-3 follow directly from the definitions.

Proposition 1. *Let $\mathcal{A}$ be any algorithm. Every execution of $\mathcal{A}$ on some instance $\mathcal{I} = \langle G, \lambda, H, k, \omega, m \rangle$ is also an execution of $\mathcal{A}$ on $\mathcal{I}' = \langle G, \lambda, H, k, \omega, m' \rangle$, where $m' \geq m$.*

Proposition 2. *If an algorithm is (k, m)-correct, then it is (k, m')-correct for all $m' \leq m$.*

Proposition 3. *Let $\mathcal{A}$ be any algorithm. Every benign execution of $\mathcal{A}$ on some instance $\mathcal{I} = \langle G, \lambda, H, k, \omega, m \rangle$, where $m \geq 2$, is also a benign execution of $\mathcal{A}$ on $\mathcal{I}' = \langle G, \lambda, H, k, \omega', 2 \rangle$, for all $\omega' \in V(G) \setminus \{H\}$.*

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. Let C_n denote an undirected ring with n nodes.

Definition 7 (Waiting). *Let $\mathcal{E}$ be an execution of an algorithm $\mathcal{A}$ on instance $\mathcal{I} = \langle G, \lambda, H, k, \omega, m\rangle$. Let W be a set of nodes that induces a connected subgraph $G(W)$ of G. We say that an agent A is* waiting *on W at time t_0 under $\mathcal{E}$ if the agent is in $G(W)$ 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 $G(W)$.*
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 $\mathcal{I} = \langle C_n, \lambda, H, k, \omega, m\rangle$ 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 traverses 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. A direct corollary of Definition 8 is the following:

Corollary 1. *If there exists an execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{I} = \langle C_n, \lambda, H, k, \omega, m\rangle$ 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 $(1, 3)$-correct or $(2, 3)$-correct algorithms has already been demonstrated in [20]. In this section, we show that no algorithm can be $(3, 3)$-correct. We achieve this by proving that, if there existed a $(3, 3)$-correct algorithm, then the adversary would be able to force one of the agents to perform a 2-traversal (Lemma 1). However, we also prove that if any agent performs a 2-traversal while executing a $(3, 3)$-correct algorithm, then the adversary can kill all three agents (Lemma 2). This establishes that a $(3, 3)$-correct algorithm cannot exist.

Lemma 1. *Let $\mathcal{A}$ be a $(3, 3)$-correct algorithm and let $\mathcal{I} = \langle C_n, \lambda, H, 3, \omega, 3\rangle$ with $n \geq 6$. There exists a benign execution of $\mathcal{A}$ on $\mathcal{I}$ under which some agent performs a 2-traversal.*

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

By Lemmas 1 and 2, the existence of a $(3,3)$-correct algorithm yields a contradiction. Therefore, we have proved the following:

Theorem 1. *There does not exist a $(3,3)$-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 $(1,5)$-correct, $(2,5)$-correct, or $(3,5)$-correct algorithms. In this section, we show that no algorithm can be $(4,5)$-correct. To this end, we first prove in Lemma 3 that, under any $(4,5)$-correct algorithm, the adversary can force some agent to perform a 3-traversal (in fact, this can even be enforced under any $(4,3)$-correct algorithm). Then, we derive a contradiction by showing in Lemma 4 that if an agent performs a 3-traversal, then four agents can die in the red hole and thus the algorithm cannot be $(4,5)$-correct.

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

Lemma 4. *Let $\mathcal{A}$ be a $(4,5)$-correct algorithm and let $\mathcal{I} = \langle C_n, \lambda, H, 4, \omega, 5\rangle$. Under any benign execution of $\mathcal{A}$ on $\mathcal{I}$, no agent performs a 3-traversal.*

By Lemmas 3 and 4 and Propositions 1 and 2, the existence of a $(4,5)$-correct algorithm yields a contradiction. Therefore, we have proved the following:

Theorem 2. *There does not exist a $(4,5)$-correct algorithm.*

4 An Optimal Algorithm for Rings with a 4-Malicious Host

In view of Theorem 1, no algorithm can achieve Periodic Data Retrieval on a ring with a 4-malicious host using only three agents (in fact, not even on a ring with a 3-malicious host). In this section, we present algorithm PDR_Rings_4-malicious, which solves the problem in the presence of a 4-malicious host in a ring, using an optimal number of four agents.

Remark 2. 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 4-malicious host.

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 if necessary.
- 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 s edges. 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.
- 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 $\langle \text{id}, + \rangle$ (resp. $\langle \text{id}, - \rangle$) 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 $\langle \text{id}, + \rangle$ and moves to v.
- A Return(+) step: The agent inserts a flag $\langle \text{id}, - \rangle$ and moves back to u.
- A Finish(+) step: The agent removes its $\langle \text{id}, + \rangle$ flag, moves to v, removes its $\langle \text{id}, - \rangle$ 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 $\langle \text{id}, + \rangle$ 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.

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 1 illustrates the high-level workings of the algorithm.

The next lemma follows by a straightforward adaptation of the proof of Theorem 1 in [20].

Lemma 5 ([20]). *Under any execution of* PDR_RINGS_4-MALICIOUS *in which not all agents are killed, Periodic Data Retrieval is achieved.*

In order to show that PDR_RINGS_4-MALICIOUS 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 7 below). For this, we take advantage of the flags left by the agents during the

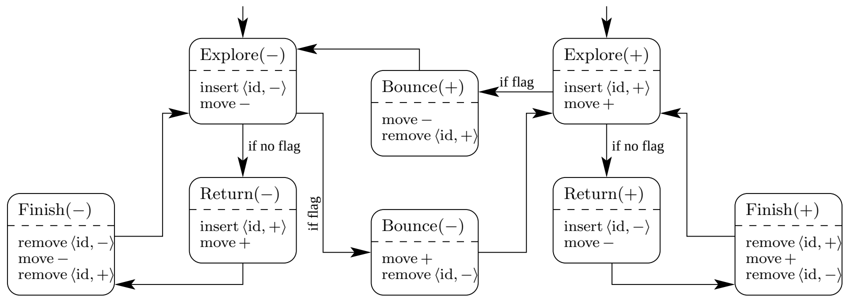

Fig. 1. A diagram of the basic operations of Algorithm PDR_RINGS_4-MALICIOUS. "move +" (resp. "move −") stands for traversing an edge in the clockwise (resp. counterclockwise) direction.

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 5, the $(4, 4)$-correctness of the algorithm follows (Theorem 3 below). The low-level operations of the algorithm play a crucial role in the proof of Theorem 3.

Definition 9. *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 4 below states an easy to check property of the algorithm.

Proposition 4. *Under any benign execution of* PDR_RINGS_4-MALICIOUS*, at most one agent can be on a given node at a given time.*

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, the only possible combinations of tags are: $\{E^+, B^-\}$, $\{E^+, F^+\}$, $\{E^-, B^+\}$, $\{E^-, F^-\}$, $\{E^+, E^-\}$, and $\{B^+, B^-\}$. Using this characterization, we can prove Lemma 6:

Lemma 6. *Under any benign execution of* PDR_RINGS_4-MALICIOUS*, it is not possible for three agents to traverse the same edge at the same time.*

Lemma 6 and Proposition 4 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 7. *For any node v other than the homebase, under any benign execution of* PDR_Rings_4-malicious*, a total of at most three agents can be in the queues of v at the same time.*

Theorem 3. PDR_Rings_4-malicious *is* $(4, 4)$*-correct in rings.*

5 An Efficient Algorithm for Rings with a Red Hole

Note that, irrespective of the number of agents, the PDR_Rings_4-malicious algorithm fails if the malicious host is a red hole. 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:

– 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 $\langle \text{id}, + \rangle$ 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. A counterclockwise agent operates completely symmetrically.

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_4-malicious, whereby the agent simply checked whether all agents previously departed from the same port had reached the destination.

As in the case of PDR_Rings_4-malicious, 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 that a straightforward adaptation of the proof of Theorem 1 in [20] yields the following:

Lemma 8. *Under any execution of* PDR_Rings_Red *in which not all agents are killed, Periodic Data Retrieval is achieved.*

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 9. *Under any execution of* PDR_Rings_Red*, at most three clockwise agents die.*

By Lemmas 8 and 9, and by the symmetric of Lemma 9 for counterclockwise agents, we obtain that PDR_Rings_Red achieves Periodic Data Retrieval with seven agents:

Theorem 4. PDR_Rings_Red *is* $(7, 5)$*-correct in rings.*

Remark 3. 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 leave a copy of it on at least one of its neighbors. Therefore, by enforcing agents to pick up data items only if they find them twice on two neighboring nodes, we ensure that an agent will never pick up a corrupted data item from the whiteboard of the red hole.

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_4-malicious and PDR_Rings_Red grows unbounded. However, it should be clear that this amount can be reduced to a reasonable function of the number of nodes and the number of agents, by deprecating and removing information which is known to be no longer useful. We defer the implementation of this mechanism to the full version of the paper.

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 behavior. Another question that remains open is what happens in other network topologies under the various malicious host models.

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)