Faulty Node Repair and Dynamically Spawned Black Hole Search

Conference Paper *in* Lecture Notes of the Institute for Computer Sciences · June 2017

DOI: 10.1007/978-3-319-59608-2_8

4 authors, including:

Wei Shi
Carleton University
98 PUBLICATIONS **584** CITATIONS
SEE PROFILE

Jean-Pierre Corriveau
Carleton University
94 PUBLICATIONS **467** CITATIONS
SEE PROFILE

William Lee Croft
Carleton University
7 PUBLICATIONS **17** CITATIONS
SEE PROFILE

Faulty Node Repair and Dynamically Spawned Black Hole Search

Wei Shi[1], Jean-Pierre Corriveau[2], and Mengfei Peng[3]

[1] School of Information Technology, Carleton University, Ottawa, Canada
wei.shi@carleton.ca
[2] School of Computer Science, Carleton University, Ottawa, Canada
jeanpier@scs.carleton.ca
[3] Faculty of Business and IT University of Ontario Institute of Technology, Oshawa, Canada mengfei.peng

Abstract. New threats to networks are constantly arising. This justifies protecting network assets and mitigating the risk associated with attacks. In a distributed environment, researchers aim, in particular, at eliminating faulty network entities. More specifically, much research has been conducted on locating a single static black hole, which is defined as a network site whose existence is known a priori and that disposes of any incoming data without leaving any trace of this occurrence. However, the prevalence of faulty nodes requires an algorithm able to a) identify faulty nodes that can be repaired without human intervention and b) locate black holes, which are taken to be faulty nodes whose repair does require human intervention. In this paper, we consider a specific attack model that involves multiple faulty nodes that can be repaired by mobile software agents, as well as a virus v that can infect a previously repaired faulty node and turn it into a black hole. We refer to the task of repairing multiple faulty nodes and pointing out the location of the black hole as the *Faulty Node Repair and Dynamically Spawned Black Hole Search*. We first analyze the attack model we put forth. We then explain a) how to identify whether a node is either 1) a safe node or 2) a repairable faulty node or 3) the black hole that has been infected by virus v during the search/repair process and, b) how to perform the correct relevant actions. These two steps constitute a complex task, which, we explain, significantly differs from the traditional *Black Hole Search*. We continue by proposing an algorithm to solve this problem in an asynchronous ring network with only one whiteboard (which resides in a node called the *homebase*). We prove the correctness of our solution and analyze its complexity by both theoretical analysis and experiment evaluation. We conclude that, using our proposed algorithm, $b + 4$ agents can repair all faulty nodes and locate the black hole infected by a virus v within finite time, when the black hole appears in the network before the last faulty node is repaired. Our algorithm works even when b, the number of faulty nodes, is unknown a priori.

Keywords: faulty node repair, black hole search, mobile agent

1 Introduction

Over the past few years, as cloud-based services have become prevalent, so has the need for effective diagnosis of all-too-frequent network anomalies and faults. As cloud servers involving multiple data centers are usually geographically dispersed (thus not physically coupled), locating a network fault physically may be expensive and difficult, if not impossible. Using software agents to locate and/or repair network faults becomes a reasonable solution and thus has attracted the attention of researchers, especially in distributed computing [31]. Many types of faults exist in a network, such as black holes (e.g., [9, 15, 19]), repairable black holes (e.g., [8, 12]), faulty agents (e.g., [5, 23]), etc. Among these, a *black hole* is a severe and pervasive problem. A black hole models a computer that is accidentally off-line or a network site in which a resident process (e.g., an unknowingly-installed virus) deletes any visiting agents or incoming data upon their arrival without leaving any observable trace [16].

In practice, many computer faults/virus cannot be completely removed by anti-virus software: After a repair, a previously infected node may still be more vulnerable than the ones that have never been infected, and can be easily re-infected. For instance, a hacker injects into a computer host a virus that can delete any incoming data and that may later be removed by an anti-virus agent. However, after repair, an unknown vulnerability remains on that host and it enables the hacker's next attack. Indeed, with fast spreading worms mentioned in [33] (such as W32/CodeRed, Linux/Slapper, W32/Blaster or Solaris/Sadmind), a host can be exploited only if the system has a vulnerability known a priori. Such virus behaviour is commonly referred to as *vulnerability dependency*. More generally, in cloud computing, the term vulnerability refers to the flaws in a system that allow an attack to be successful [25]. The vulnerability security issue has been widely discussed in research works such as [1, 6, 24].

Cooper *et al.* [8] first introduced a type of weaker black hole, which he called a *hole*, that eliminates any incoming data but can be repaired by the first encountering agent. Assuming vulnerability dependency, the hacker can then inject an even more powerful virus and turn this repaired host into a genuine (i.e., unrepairable without human intervention) black hole at some point in the future. Our work originates in that attack model. A black hole is still taken to be a node that is not repairable without human intervention. But to avoid any ambiguity around the term "hole", we will refer to a node with abnormalities that can be repaired by a software agent as a *faulty node* (rather than a hole). In this paper, we introduce the *Faulty Node Repair and Dynamically Spawned Black Hole Search problem* (*repair and search problem* for brevity).

In our new attack model, there are multiple faulty nodes. Each such node eliminates any incoming data and can be repaired upon being visited by an antivirus agent who, in effect, "dies" at the end of this repair. That is, following Cooper [8], we assume there is a cost for repairing a fault, namely, the repairing agent is unable to continue exploring the network. Furthermore, we assume that when multiple antivirus agents simultaneously enter a faulty node, they all die

at the end of the repair.[4]. We assume this worst case scenario for the design of our solution to the proposed repair and search problem. Obviously, fewer agents are required in less damaging cases.

In our attack model, a faulty node, once repaired, behaves like a safe one but remains vulnerable and can be infected again after attacked by what we call a *gray virus*. A *gray virus* (GV for brevity) is a piece of malicious software that can infect a repaired node (due to the latter's vulnerability) by residing in it and turning it into a black hole. In this paper, we consider what we call a one-stop GV, that is, a virus that permanently resides in the node it infects and thus cannot harm other nodes. (More generally, a *multi-stop gray virus* can infect multiple repaired nodes.) A GV is taken to have no destructive power on a safe node or link. Here, we consider a single one-stop GV that infects a single faulty node. That is, we consider searching for a single black hole. (More generally, there could be multiple black holes resulting from one or more multi-stop GVs.) Furthermore, in this paper, we specifically study the search and repair problem in an asynchronous ring network.

The solution we propose for this version of the repair and search problem uses a team of *mobile agents* to repair all faulty nodes and locate the single black hole (by marking the edges leading to it). These agents have limited computing capabilities and bounded storage. They all obey an identical set of behavioural rules (referred to as the "protocol"), and can move from a node to a neighbouring node. Also, these agents are anonymous (i.e., do not have distinct identifiers) and autonomous (i.e., each has its own computing and bounded memory capabilities). Such characteristics are systematically adopted for the traditional black hole location problem in computer networks.

Contrary to the traditional black hole search [32], in which all agents start in a network knowing a priori that there is one and only one back hole, in our proposed new attack model, a repaired faulty node can be infected again and turned into a black hole at any point in time (regardless of the agents traversing the network and trying to repair faulty nodes). That is, at what time a node becomes the black hole is unpredictable. Additionally, this unpredictable black hole may coexist with multiple faulty nodes. This drastically changes the nature of the black hole search problem in asynchronous networks. That is, the possible scenarios we must consider are significantly more complex than those associated with traditional black hole search. Let us briefly elaborate. To locate a black hole in traditional black hole search in an asynchronous network, there is a commonly-used technique called *cautious walk*: a first agent has to leave a "mark" indicating a potential danger (e.g., a token or a whiteboard message) in its current node before it moves along a link potentially leading to the black hole. When a second agent sees this mark, it does not go to visit the same potentially dangerous node. This technique is used to minimize the loss of mobile agents. The cautious walk technique points to the fact that the only mechanism used to terminate a traditional black hole search algorithm is to let at least one agent survive and

[4] This is the worst case scenario, which we use to calculate, later in the paper, the theoretical maximum number of agents sacrificed to solve the problem.

successfully traverse the entire network except one node. This only *unexplored* node (i.e., which has never been visited by any agent) is then declared to be the black hole. But when there are multiple faulty nodes in the network, even when more sophisticated communications between agents are available, none of the existing black hole search algorithms solve the repair and search problem. This is because, in these algorithms, there is no mechanism to distinguish a black hole from a faulty node. Consequently, given a faulty node would be treated the same way as the black hole, no agent is able to successfully explore $(n-1)$ nodes and survive.

2 Related Work

The problem of finding the most efficient solution (with respect to time and minimum number of agents required) for the black hole search is studied in an edge-labeled undirected synchronous network using 2 co-located agents using the *face-to-face* communication model in [9–11, 26, 27]. Czyzowicz *et al.* [10] show any efficient solution is NP-hard, and propose a 9.3-approximation algorithm for it. Klasing *et al.* [27] prove that this problem is not a polynomial-time approximation within any constant factor less than $\frac{389}{388}$ (unless P=NP), and give a 6-approximation algorithm. Czyzowicz *et al.* present a $\frac{5}{3}$-approximation algorithm in an arbitrary tree without a map in [9]. Furthermore, Klasing *et al.* [26] provide a $3\frac{3}{8}$-approximation algorithm for an arbitrary network with the help of a network map.

The black hole search problem in an asynchronous network is much more complex and more significant in practice. Dobrev *et al.* [20] introduce an algorithm to locate the black hole in an un-oriented ring network with dispersed agents in $O(kn + n \log n)$ moves. For some other common interconnection networks, Dobrev *et al.* [13] present a general strategy to locate the black hole in $O(n)$ moves by using 2 co-located agents. Shi *et al.* [32] prove that 2 co-located agents, each with $O(1)$ tokens, can locate the black hole in $\Theta(n)$ moves for hypercube, torus and complete networks. Moreover, for an arbitrary unknown network graph with known n, Dobrev *et al.* [14] present an algorithm using $\Delta + 1$ agents, one token per agent and $O(\Delta^2 M^2 n^7)$ moves to locate the black hole. Here, M is the total number of edges of the graph. In an arbitrary network, Dobrev *et al.* [17] prove that in the whiteboard model, the black hole search problem can be solved with $\Delta + 1$ agents in $\Theta(n^2)$ moves without network maps. Balamohan *et al.* [3] prove that in an unknown graph with a constant number of agents, at least $\Delta + 2$ agents and at least 3 tokens are necessary in total to locate the black hole, where Δ is the maximum node degree.

Multiple black hole search (MBHS for brevity) problem has been studied by Cooper *et al.* [7] in synchronous networks. Later, the same authors [8] present solutions to the multiple repairable black holes (faulty nodes) problem. D'Emidio *et al.* [12] study the same problem under the same condition as [8] with a change of one assumption: if more than one agent enters the same faulty node at the same time, all agents die. Flocchini *et al.* tackle the MBHS problem via a *subway*

model in [21]. The authors use carriers (the subway trains) to transport agents (the passengers) from node to node (subway stops), and the black holes no longer affect the carriers and can only eliminate the agents. After assuming that the graph is strongly connected after all black holes have been removed, Kosowski *et al.* [28] study a synchronous network with arbitrary size, while Flocchini *et al.* [22] study the MBHS problem with asynchronous dispersed agents.

Cai *et al.* [5] study a network decontamination problem with a black virus, which is related to both black hole search and intruder capture problems. The authors define a black virus as a dangerous process that is initially resident in the network. A black virus behaves like a moving black hole that can destroy any arriving agent and can move from node to node. However, unlike a black hole that cannot be repaired or destroyed, a black virus can be eliminated when it enters into a node with an anti-viral agent. Luccio *et al.* [30] consider a mobile agents rendezvous problem in spite of a malicious agent, which is similar to [18], which rendezvouses agents in a ring in spite of a black hole. While a malicious agent in [30] can only block other agents from visiting its resident node and can move in the network at arbitrary speed, a black hole in [30] can delete all visiting agents but it cannot move. Královič *et al.* [29] research a periodic data retrieval problem using a whiteboard in asynchronous ring networks with a malicious host. The malicious host can manipulate the agent by storing and copying it and releasing the replica later to confuse other agents, or by killing an agent. Bampas *et al.* [4] improve this result by showing that at least 4 agents are required when the malicious host is a gray hole, which can choose to behave as a black hole or as a safe node, and 5 agents are necessary when the whiteboard on the malicious host is unreliable.

3 Premises

In this section, we present our assumptions for the solution we propose for the *Faulty Node Repair and Dynamically Spawned Black Hole Search problem* in an asynchronous ring network.

Let $G = (E, V)$ denote an edge-labeled undirected ring network, where E is the set of edges, V is the set of network nodes and n ($n = |V|$) denotes the number of nodes in G. $(u, v) \in E$ represents the link from u to v, where $u \in V$ and $v \in V$ and u to v are neighbouring nodes. The links and nodes in the network enforce a FIFO rule, that is, mobile agents cannot overtake each other when traveling in the same direction over the same link or node. Without this assumption, systematic termination of a repair and search algorithm with minimal number of agents cannot be guaranteed.

Let $\mathcal{A}$ denote a group of k ($k \geq 2$) identical mobile agents initially waking up in the same node referred to as their *homebase (hb)*. This *homebase* is assumed to be safe in the ring network: it is neither faulty nor a black hole. These agents have limited computing capabilities and bounded storage[5], obey the same set

[5] That is, minimal storage just sufficient to keep track of the number of moves an agent has performed during each exploration of a new node.

of behavioural rules (the "protocol"), and can move from node to node via neighbouring nodes. We make no assumptions on the amount of time required by an agent's actions (e.g., computation or movement, etc.) except that it is finite. Thus, the agents are *asynchronous*. Also, these agents are anonymous (i.e., have no ID) and know the topology of the network in which they reside. Most importantly, these agents have no knowledge of the number of faulty nodes. We let $V_f \subseteq V$ denote the set of b ($b < n$) faulty nodes (the homebase being, by design, free of fault). Most importantly, in this paper we postulate V_f is not dynamic: contrary to the black hole, faulty nodes are not dynamically spawned but all already exist at the start of the algorithm. Once a faulty node has been repaired, it is referred to as a *repaired node*. We emphasize that, unlike a safe node, a repaired node can be infected by a GV and turn into a black hole.

We postulate that a *whiteboard* [16] (i.e., shared memory) in the hb offers the *only* means of communication between agents. This whiteboard in hb can be accessed by agents in *fair mutual exclusion* [2].

We assume the network is an un-oriented ring, that is, there is no agreement on a common *sense of direction* among the agents [16]. However, using the whiteboard in hb, all agents shall be able to agree on what corresponds to the clockwise direction (also referred to as the left direction) and the counter-clockwise direction (also referred to as the right direction) of the ring. In order to ease the understanding of our algorithm's description, N_0, N_1, ... N_{n-1} are used to label the nodes of the ring sequentially using the left direction starting from the hb. Such labelling is only used to improve the understandability of our explanations and proof; it is not required by our algorithm per se.

Observation 1 *When a repaired node gets reinfected by a GV only after all faulty nodes have been repaired, the Repair and Search problem becomes a faulty node repair problem followed by a dynamic single black hole search problem for which all the possible locations of the black hole are known a priori since only repaired nodes can be reinfected.*

In the rest of this paper, we are first and foremost interested in studying the more complex scenario in which a one-stop GV may infect a repaired node before the last faulty node is repaired. It is the coexistence in the network of a black hole with at least one faulty node that makes the *Repair and Search* problem complex. We will provide a solution for this challenging scenario, present a theoretical proof of correctness and complexity analysis for it, as well as discuss its simulation. The simpler scenario in which the black hole appears only after all faulty nodes are repaired is specifically discussed in Section 7.

4 Algorithm and Solutions

4.1 General Description

We postulate that the status of each node in a network can be either "unexplored" or "faulty" or "repaired" or "black hole" or "safe" or "unknown".

Furthermore, we define the general goal of each agent to consist in exploring a new node (which we also call an *unexplored* node) and updating the whiteboard upon returning to the *hb*. During this exploration, an agent may die after repairing a faulty node, or in a black hole or survive and successfully return to the *hb* and restart the procedure of exploring a new node. This "new exploration/update whiteboard" task gets repeated until the status of each node is entered (or equivalently, marked) as either a repaired node, or a black hole or a safe node. In order to prevent multiple agents dying in the same faulty node or black hole, we develop a status marking process as part of the protocol agents execute. The following paragraphs, as well as Tables 1 and 2, explain this process:

When a first agent A wakes up at *hb*, it initializes the whiteboard as shown in Table 1. All nodes are initially *unknown* nodes. Agent A then puts a *leaving* mark (?) in the cell of First Agent for node N_1, and goes to visit node N_1 going left, which is recorded as (l) after its ? mark. After visiting node N_1, agent A immediately attempts to return to *hb*. If and when A returns to *hb*, agent A changes its leaving mark to a *returned* mark ($\surd$) (Table 2 shows examples).

Table 1. Homebase Whiteboard Initial State

Node List	First Agent	Second Agent	Third Agent	Fourth Agent	Repaired Node List
N_1					
...					
N_{i-1}					
N_i					
...					
N_{n-j}					
N_{n-j+1}					
...					
N_{n-1}					

Table 2. An example of how agents indicate their status.

Node List	First Agent	Second Agent
N_1	$\surd(l)$	
N_2	$\times$	$\surd(l)$
...		
N_i	$?(l)$	$?(l)$
...		
N_{n-2}	$?(r)$	
N_{n-1}	$\surd(r)$	$?(r)$

By repeating this process, agent A explores nodes N_2, N_3, ..., N_i. Other agents, such as B, may wake up any time during A's explorations. B then searches the whiteboard starting at the top row. When B sees a leaving mark for node N_i, agent B attempts to go to node N_i to confirm the status of that

node. B puts a leaving mark in the Second Agent column for node N_i. After visiting node N_i, agent B immediately attempts to return to hb and, if successful, changes its leaving mark into a returned mark. By the time agent B returns, agent A may have returned (i.e., A's mark has changed from leaving to returned), which entails node N_i is not a faulty node. Otherwise, because of the FIFO rule, agent B concludes that agent A has died after repairing faulty node N_i. (If B made it back to hb, then neither A or B encountered the black hole.) In this situation, B will change the *leaving* mark (?) of A into a *died* mark ($\times$) and mark N_i as a repaired node under the Repaired Node List column (see Scenario S5 in Table 3).

Table 3. Scenarios for agents leaving and returning to the homebase, as marked on the whiteboard.

Scenarios	First Agent	Second Agent	Targeted node is	No. of agents of status unknown
S1	? (l/r)		Unknown	1
S2	$\surd$ (l/r)		Safe	0
S3	? (l/r)	? (l/r)	Unknown	2
S4	$\surd$ (l/r)	? (l/r)	Safe	1
S5	$\times$	$\surd$ (l/r)	Repaired node	0 (1 died)
S6	$\surd$ (l/r)	$\surd$ (l/r)	Safe	0

?: an agent of status unknown, that left to explore a node.
$\times$: an agent that died either in a black hole or after repairing a faulty node.
$\surd$: an agent that has returned to hb.
(l/r): left or right; each agent actually using only one of these 2 directions at any point in time.

While agents A and B are out exploring in the left direction, agents C and D may wake up. C and D immediately start exploring the ring in the right direction ($?(r)$) to visit nodes N_{n-1}, N_{n-2}, ..., N_{n-j}. This mechanism is designed to avoid unnecessary loss of agents (i.e., if the black hole has just appeared, sending more agents in the same direction as A and B will lead to agent loss). That is, if 2 agents are already exploring in one direction, a newly awake agent will start exploring the ring in the opposite direction. Furthermore, as long as the ring still has at least one unknown node and 4 agents are currently exploring such nodes, a newly awake up agent will just wait at hb until at least 1 of these 4 agents returns. This mechanism is used to minimize the total agent moves, that is, to minimize the network traffic. The details are described in Procedure *New Node Exploration* (subsection 4.2).

When an agent notices that there is only one unknown node left in the network, it starts executing Procedure *Find the Meeting Node*. Eventually 2 agents enter the last unknown node from the left direction and 2 agents from the right one. If one of these 4 agents dies in the black hole (that would have just appeared) on its way to check the last unknown node, this last unknown node is not a black hole. Hence, at least 1 out of the 4 agents left to explore

the last unknown node can return to the *hb* successfully. If the last unexplored node is the black hole, we need a mechanism to make sure at least one agent is able to safely return to the *hb* and somehow conclude that the last unexplored node is indeed the black hole. This mechanism is described in Procedure *Find the Meeting Node* (subsection 4.3) and Procedure *Double Check* (algorithm 3).

4.2 Procedure New Node Exploration

Whenever an agent returns to the *hb*, it looks though the node list of the whiteboard from the top to the bottom. The agent may find a node to be: *unexplored*, that is, a node that has never been visited by any agent, in which case there is no mark on the whiteboard (i.e., the row of the node in Table 1 is empty); or *repaired*, that is, a node that has a $\surd$ under the Second Agent column and a $\times$ mark under the First Agent column (i.e., the Second Agent returned but the first one did not. See Scenario S5 in Table 3); or *safe*, that is, a non-faulty node that has a $\surd$ under the First Agent column (i.e., the First Agent has returned; see Scenarios S2, S4 and S6 in Table 3); or *unknown*, that is, a node that has a ? under the First Agent column or both First and Second Agent columns (i.e., both agents have left but no agent ever returned. See Scenarios S1 and S3 in Table 3). The status of a node is considered to be *known* if it is either *safe* or *repaired*.

While going through the nodes list, if there is any unexplored node, then an agent A counts pd_l, the number of $?(l)$. It determines its next step accordingly: If A cannot find an unexplored node, A will finish searching the whole list and execute Procedure *Find The Meeting Node*. When at most one agent has left in the left direction ($pd_l < 2$), agent A leaves in the left direction to visit an unexplored node or confirm the status of that node. When $pd_l = 2$, the agent counts pd_r, that is, the number of $?(r)$. If $pd_r < 2$, agent A leaves in the right direction. When 2 agents are out in each direction, agent A waits at *hb* until at least one agent returns. When node N_i is the last unexplored node in the network, it becomes a *meeting node*.

4.3 Procedure Find the Meeting Node

Agent A executes Procedure *Double Check* (see next subsection) if there are no more unknown nodes left in the network. Otherwise, A counts pd, the number of agents of status unknown, (or equivalently, status unknown agents) in the entire list and executes the following accordingly:

1. When $pd > 4$, A waits at *hb*.
2. When $pd = 4$ and the 4 status unknown agents are not exploring the same node (i.e. all 4 left ? mark on the same node), A waits at *hb*.
3. When $pd = 4$ and the 4 status unknown agents have marks for the same node, A starts Procedure *Double Check* immediately.
4. When $pd < 4$ and there are no nodes in Scenario S1: a. If all status unknown agents are not on the same node, A waits at *hb*; b. If all status unknown agents are on the same node, A goes to that node.

Algorithm 1 New Node Exploration

```
1: initialize the whiteboard to Table 1
2: loop
3:     if an unexplored node N_i is found then
4:         count the number pd_l of status unknown agents out in the left direction ?(l)
5:     else if no unexplored node is found then
6:         execute FIND THE MEETING NODE
7:     end if
8:     if pd_l = 0 then
9:         go to node N_i
10:    else if pd_l = 1 and node N_{i-1} is in Scenario S1 of Table 3 then
11:        go to node N_{i-1}
12:    else if pd_l = 1 and node N_{i-1} is in Scenario S4 of Table 3 then
13:        go to node N_i
14:    else if pd = 2 then
15:        count the number pd_r of ?(r)
16:        leave in the right direction when pd_r < 2, otherwise wait at hb
17:    end if
18:    upon arriving to target node, return to hb immediately, then if successful change
           own ?(l/r) into √(l/r)
19:    if the current agent is the Second Agent and the First Agent is ? then
20:        change the ? of the First Agent to ×
21:    end if
22: end loop
```

5. When $pd < 4$ and there are one or two nodes in Scenario S1, and there is an unexplored node X between hb and these nodes on either direction, A goes to X. Otherwise, A waits at hb until one returns.

Upon return to the hb from a node with 4 status unknown agents, A changes the mark left in third agent column into $\times$.

4.4 Procedure Double Check

As detailed in Procedure Find the Meeting Node, an agent A will only start executing Procedure Double Check when A sees that either all nodes' statuses are known (either safe or repaired) or all but one nodes' statuses are known (i.e., 4?s are on that last node).

Agent A first mark all repaired nodes in Table 1. Then it decides its next action to take based on the following situations (according to the current marks in the Third Agent column):

1. If there are already 2 status unknown agents performing Double Check (2 ?s in this column), A waits at hb until one of these two agents returns;
2. If there is only 1 status unknown agent currently performing Double Check (only 1? in this column), A searches this Third Agent column from top to bottom until it finds an empty cell. If the empty cell is above the status unknown agent, A checks that node from the left direction (i.e., puts a $?(l)$ in the cell, then goes to that node). Otherwise A checks the first repaired

Algorithm 2 FIND THE MEETING NODE

```
1: loop
2:     count the number of status unknown agents pd
3:     if every node is known to be safe or repaired then
4:         execute Double Check
5:     end if
6:     if pd > 4 or pd = 4 and the 4 ? are not on the same node then
7:         wait at hb
8:     else if pd = 4 and the 4 status unknown agents are on the same node then
9:         execute Double Check
10:    else if pd < 4 and no node is in Scenario S1 then
11:        if all status unknown agents are not on the same node then
12:            wait at hb
13:        else if all status unknown agents are on the same node then
14:            go to this node, from the direction in which there is less than 2 agents
15:        end if
16:    else if pd < 4 and one/more than one node is in Scenario S1 then
17:        go to a reachable node in Scenario S1, otherwise wait at hb
18:    end if
19:    upon arriving, return to hb
20:    if hb is reached and the cell of the Third Agent for the same node is ? then
21:        change the third ?(l/r) into ×
22:    end if
23: end loop
```

node on the right side of hb (leaves in the right direction after putting down a $?(r)$). A returns to hb immediately after visiting this target node. Upon returning to hb, it changes the ? to a $\surd$.

3. If there is only 1 status unknown agent (only 1? in this column) and all but one repaired nodes have been checked (all but one cell in the Third Agent column are marked with $\surd$), the node marked by that status unknown agent is the black hole.

5 Theoretical Correctness and Complexity Analysis

Lemma 1. *There can be no more than* 4 *status unknown agents co-existing in the network as long as at least one node has not been marked on the whiteboard by any agent. At least* 1 *of these* 4 *agents will return to hb.*

Proof. In the homebase hb, as long as an agent A can find an unexplored node in the node list, it always needs to explore a new node (by executing Procedure New Node Exploration) before it executes any other procedure.

An agent A always searches the node list starting from the top first if at most one agent has left in the left direction, in which case A will also leave in the left direction. Otherwise it searches the node list starting from the bottom. Hence, there will never be more than 2 status unknown agents leaving in the left direction. Similarly, when A searches from bottom to top of the node list, A

Algorithm 3 DOUBLE CHECK

```
1: search the repaired node list
2: if the list is blank then
3:     mark all repaired nodes in the list
4: end if
5: while the black hole has not been located do
6:     search the Third Agent column
7:     if there are 2 ? in this column then
8:         wait at hb until an agent returns
9:     else if there is 1 ? in this column then
10:        search this column from top to bottom until an empty cell of a repaired
               node is found
11:        if the empty cell is above the ? then
12:            go left to that node, upon arriving, return to the hb immediately
13:        else if the empty cell is below the ? then
14:            search this column from bottom to top until an empty cell of a repaired
                   node is found, go right to that node, upon arriving, return to the hb
                   immediately
15:        else if an empty cell cannot be found then
16:            the black hole is determined to be the node with ? mark
17:            ALGORITHM TERMINATES
18:        end if
19:    else if there is no ? in this column then
20:        search this column from top to bottom until an empty cell of a repaired
               node is found, go left to that node, upon arriving, return to the hb
               immediately
21:    end if
22:    upon arriving at hb, change its ? into a √
23: end while
```

leaves in the right direction if at most one agent has left in the right direction. If A finds 2 agents have left in both left and right directions, A will wait at hb until Table 1 is changed by a returned agent (see Line 16 in Procedure New Node Exploration). Consequently, there will never be an occasion in which any agent will leave hb when there are two ?s on each side of it. Hence, there cannot be more than 4 status unknown agents as long as at least one node is unexplored.

We now prove that at least 1 of these 4 agents will return to hb eventually. It is trivial to observe that all the explored nodes are in one or two consecutive sections in a ring: when there is no unexplored node remaining in the ring, all explored nodes are in one consecutive section; otherwise, the two sections of explored nodes are separated at each end by the hb and a consecutive section of unexplored nodes. We call these two sections the left part and the right part. When there are 4 status unknown agents in the network, it can only be the case that 2 are in the left part and 2 in the right part. According to our assumptions, we know that once the black hole appears, it can only exist either in the left part, or in the right part.

Clearly if the black hole has not appeared yet, the two second-agents (1 on each side) in both parts will return to *hb* traversing through the section of the ring with consecutive explored nodes while the two first-agents (1 on each side) may die if the Last unexplored node happens to be a faulty node. If the black hole appears in the left part, the second-agent in the right part will return successfully and the two agents in the left part die in the black hole. Similarly, if the black hole is in the right part, the second-agent in the left part will return while the other three die. In summary, no matter when the black hole appears and no matter where the black hole is, in the process of exploring the Last an unexplored node, at least 1 of the 4 status unknown agents will return to the *hb*.

Lemma 2. *At most* 5 *status unknown agents coexist during the time that the last node in the network is being explored. At least* 1 *of these* 5 *status unknown agents will return to hb.*

Proof. When the last unexplored node is being explored, according to Procedure New Node Exploration line 16, only when there are fewer than 4 status unknown agents coexisting in the network, a newly waking up agent will decide accordingly to go to the last unexplored node. Furthermore, according to Procedure Find The Meeting Node lines 4 and 7, Procedure Double Check can be executed when either all nodes' statuses are known or when 4 status unknown agents are exploring the same node (i.e., the last one in the network). This latter case is where the fifth agent is needed in the network. Since it is possible, this last node is a black hole, in which all 4 agents die. In all other cases, a newly waking up agent waits at *hb*.

If this last node is the black hole, none of the 4 agents can return. According to Procedure Double Check a new agent (the 5^{th}) enters the network. As the fifth status unknown agent in the network it goes to check all but the last node. It concludes that the last node is the black hole according to Line 10 in Procedure Double Check. If this last node is not a black hole, the fifth Agent may die stepping into a black hole that just appeared. However, according to Lemma 1, at least one of these other 4 agents can successfully explore the last unexplored node and return to the *hb* successfully. Eventually one of the two will die in the black hole while the last one of these 4 survives. Therefore, at least 1 of these 5 status unknown agents will return to *hb*.

Lemma 3. *All faulty nodes will be repaired within finite time.*

Proof. If a faulty node N_x has not been repaired, its status shown in the whiteboard in *hb* must be either unexplored or unknown, that is, the exploring agent either died after repairing a faulty node or in a back hole or has not returned to *hb* yet. If N_x is unexplored, according to Lines 4, 9 and 11 in Procedure New Node Exploration, an agent will explore N_x and any other unexplored node before it executes the procedure that can lead to the termination of the algorithm.

If N_x is a status unknown node, it can be either in Scenario S1 or S3. When N_x is in Scenario S1, according to Lines 11 and 13 in Procedure New Node Exploration, it is either the case that the First Agent returns to *hb* after exploring

N_x and marks this node safe on the whiteboard; or a Second Agent will explore N_x and consequently change the marking on the whiteboard into Scenario S3.

When N_x is in Scenario S3, it may become S4-safe, S5-repaired, S6-safe, or stay S3-unknown. As proven in Lemma 1, at most 2 nodes may be in Scenario S3. When 2 nodes are in Scenario S3, at least one agent will return to *hb*, since there is only one black hole. This returning agent will change one of the two Scenario S3 nodes. Consequently, at most 1 node remains in Scenario S3.

For this last unknown node, a third and a fourth agent will go to this node according to Line 14 in Procedure Find the Meeting Node. As proven in Lemma 2, as long as this node is not a black hole, one of the 4 agents will return to *hb*. If this node is the black hole, it must have been a repaired node first. Therefore, we conclude that all faulty nodes will be repaired within finite time.

Lemma 4. *Procedure Double Check locates the black hole correctly.*

Proof. Procedure Double Check gets executed in only two situations: 1) all nodes have known status, 2) only one node is unknown and it has 4 status unknown agents exploring it. In the former case, according to our initial assumption, the black hole has already appeared. According to Line 20 in Procedure Double Check, each new agent or a newly returned (to *hb*) agent simply leaves to check each node one by one, and the last repaired node that has no agent returned is the black hole. In the latter case, a Fifth Agent is needed to continue the Double Check. As previously proven in Lemma 2, at least 1 of these 5 status unknown agents will return to *hb*. If the returning agent is this Fifth Agent, it will continue checking another node until it returns to *hb* and notices that there is only one repaired node with no agent that has returned. If the returning agent is one of the 4 agents that were marked on the last status unknown node, according to Line 21 in Procedure Find the Meeting Node, the status of this unknown node becomes known. Consequently, this latter case becomes the first case. Therefore the black hole is located.

Lemma 5. $b + 4$ *agents suffice to repair all faulty nodes and locate the black hole in a ring network using only one whiteboard in the homebase.*

Proof. To repair $b - 1$ faulty nodes, $b - 1$ agents are necessary and sufficient. In the worst case, the last unknown node is the black hole and all 4 status unknown agents die in it, and one more agent is needed to perform the Procedure Double Check. All other cases are proven in Lemma 2: at least 1 of these 5 agents will return to *hb* and locate the black hole. Therefore, $b + 4$ agents suffice.

Lemma 6. *All faulty nodes can be repaired and the black hole can be located within* $O(n^2)$ *moves.*

Proof. In the worst case, the b faulty nodes are the nodes from N_{n-1} to N_{n-b} and each node in the ring has been visited by 2 agents in Procedure New Node Exploration. Therefore, it costs $2*2*(1+2+3+4+\cdots+(n-1)) = 2(n-1)(n-2)$ moves. The last unknown node may be explored by 4 agents in Procedure Find the Meeting Node. Hence, at most $4*2(n-1)$ moves are performed. In Procedure

Double Check, each node needs to be visited again, which costs $2(n-1)(n-2)$ moves. In total, $4*(n-1)(n-2)+4*2(n-1)=O(n^2)$ moves are needed.

Theorem 1. *Algorithm Dynamically Spawned Black Hole Search can repair all faulty nodes and locate the black hole with $b+4$ co-located agents in $O(n^2)$ moves using only one whiteboard in the homebase when the black hole appears in the network before the last faulty node is repaired.*

6 Verifying Correctness and Complexity using Simulation

In this section, we present the experimental results obtained from a series of Java simulations of the proposed algorithm. The experiment is done in a ring network with only one whiteboard in the homebase node, which can only be accessed when the agents are in the homebase. All agents start from this homebase and execute the same protocol as described in the previous sections. We use the variable $faulty\_posb$ to capture the percentage of faulty nodes in the experimental network. Its value ranges from 20% to 40%. Furthermore, in order to simulate a black hole dynamically-spawned by the GV, each repaired node is assigned a probability of becoming this black hole.

At the beginning of the exploration, the agents do not know the number of faulty nodes or their locations.

To make the simulation more realistic, the distance of each link between two neighbouring nodes are randomly assigned to simulate an asynchronous network; that is, the time an agent spends on a link is unpredictable but finite. Furthermore, the implementation has a task scheduler that will wake up a sleeping agent after a random amount of time. This is used to simulate the behaviour of agents that sleep an unpredictable amount of time in an asynchronous network.

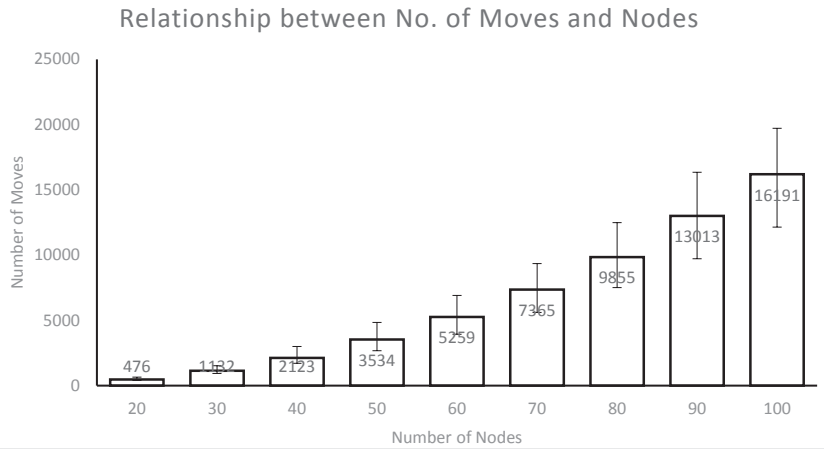

Fig. 1. The Relationship between Number of Moves and Number of Nodes

Our simulation is executed in networks varying from 20 to 100 nodes. The execution of a simulation is considered to be successful if the location of the black hole and faulty nodes are correctly marked on the homebase whiteboard. Otherwise, the simulation is counted as a failure. For each successful simulation, we count the total number of moves that are used to repair all faulty nodes and locate the black hole. For $faulty\_posb = 20\%, 30\%, 40\%$ and $n = 20, 30, 40, ..., 90, 100$, we provide 100 independent successful runs, for a total of 2700 runs. We report that 100 executions were required in order to obtain 100 independent successful runs. In other words, no failures were observed for any of the settings we tried. The results show that $b + 4$ agents are sufficient to finish the repair and search task. Additionally, the test results show that in 14.8% of the runs the task can be finished using only $b + 3$ agents or fewer.

Figure 1 reports on our results for the average number of moves, and displays the lower and upper bound for the total number of moves for each setting. These results confirm that $O(n^2)$ moves suffice to repair all faulty nodes and locate the black hole in all simulations. Clearly, (as confirmed in Figure 1) the larger the network, the more moves are necessary for the task to complete.

We further analyze whether the number of faulty nodes affects the number of moves. Figures 2 and 3 show that as the number of faulty nodes increases, the total number of moves also has a slight increase. However, this increase is not regular. Thus, we conclude the number of faulty nodes does not appear to be directly correlated to the total number of moves performed by the team of agents.

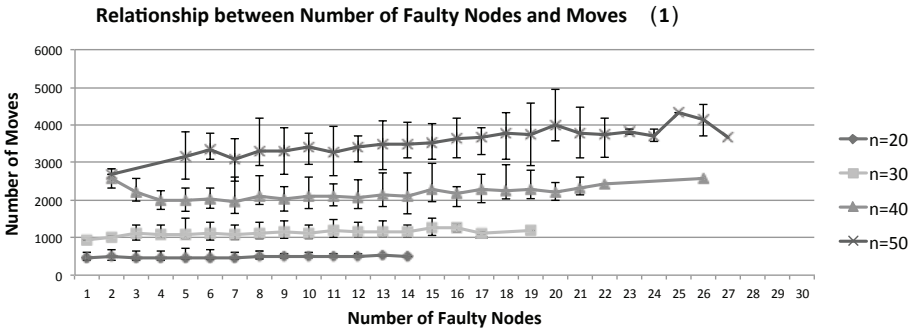

Fig. 2. The Relationship between Number of Moves and Faulty Nodes (20 to 50-node networks)

In summary, the theoretical analysis and simulation results both prove that all faulty nodes can be repaired and the black hole can be located with $b + 4$ agents in $O(n^2)$ moves using only one whiteboard in the homebase, with no a priori knowledge of b, the number of faulty nodes.

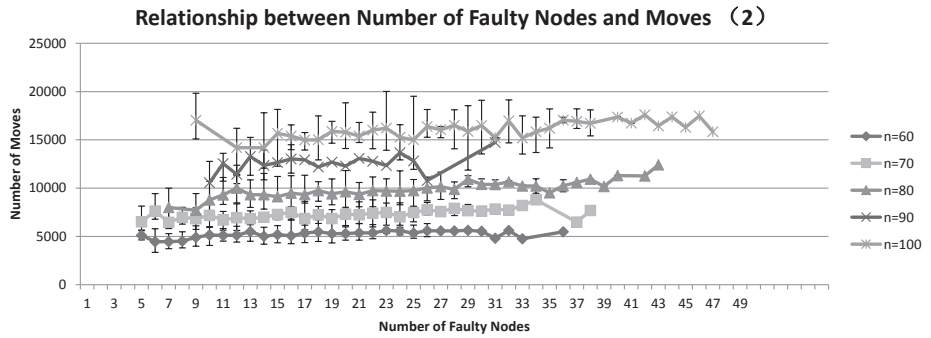

Fig. 3. The Relationship between Number of Moves and Faulty Nodes (60 to 100-node networks)

7 Discussion on the Dynamic Nature of the Black Hole

In this section, we discuss the special scenario mentioned in Observation 1, that is, when the black hole only shows up after all faulty nodes have been repaired. This is a simpler case since the faulty node repair and black hole search problem that we defined in this paper now becomes a multiple faulty nodes repair followed by a dynamic black hole search. Executing procedure New Node Exploration is enough to have all the faulty nodes repaired. The dynamic black hole search can then be dealt with. It is trivial to illustrate the difference between such a dynamic black hole search and the traditional black hole search: In traditional black hole search, the black hole is assumed to exist before any search algorithm begins. As a result, an agent dies in a black hole after leaving its *hb* as shown on the left in Figure 4. In contrast, in a dynamic black hole search, a repaired node may still be a non-black hole the first time an agent visits it after leaving its *hb*. Then, because the exact time at which the one-hop GV turns a repaired node into a black hole is unpredictable but finite, an agent may die in a dynamically spawned black hole after checking on another node further down in the network (e.g., counter clockwise leaving from *hb*.), that is, while returning to its *hb*, as shown in the right figure in Figure 4.

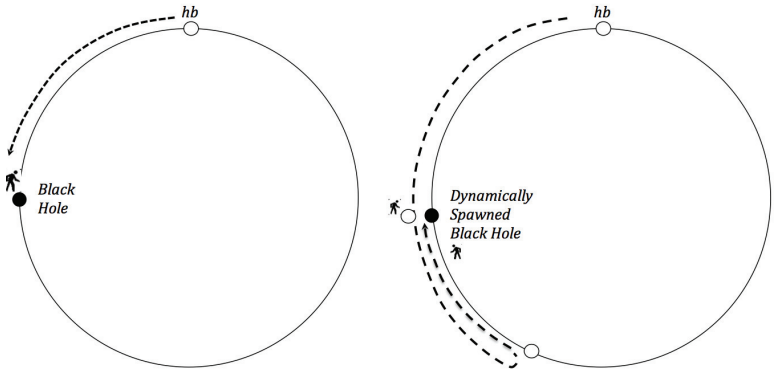

Fig. 4. Difference between searching a Dynamic Spawned Black Hole Search vs. a Static Black Hole

Once the black hole shows up in the network, the problem becomes the same as for the traditional (i.e., static) black hole search problem. Consequently, in this scenario, locating the dynamically spawned black hole comes at the cost of 1 more agent than the cost of solutions to the traditional black hole problem.

8 Conclusion and Future Work

In this paper, we first present a new attack model involving both faulty nodes and a gray virus (GV) that may infect a repaired faulty node at an arbitrary point in time, turning it into a black hole. We then propose a solution to the *Faulty Node Repair and Dynamically Spawned Black Hole Search problem* with only one whiteboard in an asynchronous ring network with the presence of a GV. Due to the possible coexistence in the network of faulty nodes with the black hole, the problem we consider is significantly more complex than the traditional black hole search (which deals with a single static black hole whose existence must be known before the search starts). We present proofs for algorithm correctness and complexity analysis and confirm these using extensive simulations. We conclude that $b + 4$ agents can repair all faulty nodes as well as locate the black hole that is created by a one-stop GV, when the black hole appears in the network before the last faulty node is repaired. We then discuss the scenario in which the black hole appears in the network after all faulty nodes have been repaired and point out the difference between this scenario and the traditional static black hole search. In this special case, we remark that repairing the faulty nodes is less complex than when allowing the black hole and the faulty nodes to coexist. Also, for this special case, searching for the dynamically spawned black hole requires merely one more agent than solutions to the traditional static black hole search.

A GV that could move from node to node and thus infect multiple repaired nodes is not discussed in this paper but left as future work. It is important to notice that in an asynchronous network, the GV may move much faster than the agents. Consequently, from the agents' viewpoint, all the repaired nodes may appear to be black holes. That is, in the case of a multi-stop GV, the *Repair and Search* problem becomes a multiple black hole search problem and thus remains unsolvable in an asynchronous network. We are currently exploring whether requiring that a multi-stop GV have to kill at least one agent before being enabled to move could enable a solution for handling a multi-stop GV in an asynchronous network.

Acknowledgments

The authors gratefully acknowledge financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant No. GPIN-2015-05390.

References

1. Almorsy, M., Grundy, J., and Müller, I. An analysis of the cloud computing security problem. In *Proceedings of APSEC 2010 Cloud Workshop, Sydney, Australia, 30th Nov* (2010).
2. Anderson, J. H., Kim, Y.-J., and Herman, T. Shared-memory mutual exclusion: major research trends since 1986. *Distributed Computing 16*, 2-3 (2003), 75–110.
3. Balamohan, B., Dobrev, S., Flocchini, P., and Santoro, N. Exploring an unknown dangerous graph with a constant number of tokens. *Theor. Comput. Sci.* (2014).
4. Bampas, E., Leonardos, N., Markou, E., Pagourtzis, A., and Petrolia, M. Improved periodic data retrieval in asynchronous rings with a faulty host. In *Structural Information and Commun. Complexity*. Springer, 2014, pp. 355–370.
5. Cai, J., Flocchini, P., and Santoro, N. Network decontamination from a black virus. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International* (2013), IEEE, pp. 696–705.
6. Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R., and Molina, J. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security* (2009), ACM, pp. 85–90.
7. Cooper, C., Klasing, R., and Radzik, T. Searching for black-hole faults in a network using multiple agents. In *Proc. of the 10th Int. Conference on Principles of Distributed Systems* (2006), OPODIS'06, Springer-Verlag, pp. 320–332.
8. Cooper, C., Klasing, R., and Radzik, T. Locating and repairing faults in a network with mobile agents. *Theor. Comput. Sci. 411*, 14-15 (2010), 1638–1647.
9. Czyzowicz, J., Kowalski, D., Markou, E., and Pelc, A. Searching for a black hole in tree networks. In *Proceedings of the 8th International Conference on Principles of Distributed Systems* (2005), OPODIS'04, Springer-Verlag, pp. 67–80.
10. Czyzowicz, J., Kowalski, D., Markou, E., and Pelc, A. Complexity of searching for a black hole. *Fundamenta Informaticae 71*, 2,3 (Aug. 2006), 229–242.
11. Czyzowicz, J., Kowalski, D., Markou, E., and Pelc, A. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability & Computing 16*, 4 (July 2007), 595–619.
12. D'Emidio, M., Frigioni, D., and Navarra, A. Exploring and making safe dangerous networks using mobile entities. In *Ad-hoc, Mobile, and Wireless Network*. Springer, 2013, pp. 136–147.
13. Dobrev, S., Flocchini, P., Kralovic, R., Prencipe, G., Ruzicka, P., and Santoro, N. Black hole search by mobile agents in hypercubes and related networks. In *OPODIS* (2002), vol. 3, pp. 169–180.
14. Dobrev, S., Flocchini, P., Královič, R., and Santoro, N. Exploring an unknown graph to locate a black hole using tokens. In *Fourth IFIP International Conference on Theor. Comput. Sci.-TCS 2006* (2006), Springer, pp. 131–150.
15. Dobrev, S., Flocchini, P., Královič, R., and Santoro, N. Exploring an unknown dangerous graph using tokens. *Theor. Comput. Sci. 472* (2013), 28–45.
16. Dobrev, S., Flocchini, P., Prencipe, G., and Santoro, N. Mobile search for a black hole in an anonymous ring. In *Proc. of the 15th Int. Conf. on Distributed Computing* (London, UK, UK, 2001), DISC '01, Springer-Verlag, pp. 166–179.
17. Dobrev, S., Flocchini, P., Prencipe, G., and Santoro, N. Searching for a black hole in arbitrary networks: Optimal mobile agent protocols. In *Proceedings of*

the Twenty-first Annual Symposium on Principles of Distributed Computing (New York, NY, USA, 2002), PODC '02, ACM, pp. 153–162.

18. Dobrev, S., Flocchini, P., Prencipe, G., and Santoro, N. Multiple agents rendezvous in a ring in spite of a black hole. In *Proceedings of the 7th International Conference on Principles of Distributed Systems* (2004), Springer, pp. 34–46.
19. Dobrev, S., Flocchini, P., and Santoro, N. Improved bounds for optimal black hole search with a network map. In *Proc. of the 11th Int. Colloquium on Structural Info. and Communication Complexity* (2004), Springer, pp. 111–122.
20. Dobrev, S., Santoro, N., and Shi, W. Locating a black hole in an un-oriented ring using tokens: The case of scattered agents. In *The 13th International Euro-Par Conference European Conference on Parallel and Distributed Computing (Euro-Par 2007)* (2007), Springer, pp. 608–617.
21. Flocchini, P., Kellett, M., Mason, P. C., and Santoro, N. Mapping an unfriendly subway system. In *Proceedings of the 5th International Conference on Fun with Algorithms* (2010), Springer, pp. 190–201.
22. Flocchini, P., Kellett, M., Mason, P. C., and Santoro, N. Fault-tolerant exploration of an unknown dangerous graph by scattered agents. In *Proceedings of the 14th International Conference on Stabilization, Safety, and Security of Distributed Systems* (2012), SSS'12, Springer, pp. 299–313.
23. Flocchini, P., Luccio, F. L., and Song, L. X. Size optimal strategies for capturing an intruder in mesh networks. In *Communications in Computing* (2005), pp. 200–206.
24. Grobauer, B., Walloschek, T., and Stocker, E. Understanding cloud computing vulnerabilities. *Security & privacy, IEEE 9*, 2 (2011), 50–57.
25. Hashizume, K., Rosado, D. G., Fernández-Medina, E., and Fernandez, E. B. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications 4*, 1 (2013), 1–13.
26. Klasing, R., Markou, E., Radzik, T., and Sarracco, F. Hardness and approximation results for black hole search in arbitrary networks. *Theor. Comput. Sci. 384*, 2 (2007), 201–221.
27. Klasing, R., Markou, E., Radzik, T., and Sarracco, F. Approximation bounds for black hole search problems. *Networks 52*, 4 (2008), 216–226.
28. Kosowski, A., Navarra, A., and Pinotti, C. M. Synchronization helps robots to detect black holes in directed graphs. In *Proceedings of the 13th International Conference on Principles of Distributed Systems* (2009), Springer, pp. 86–98.
29. Královič, R., and Miklík, S. Periodic data retrieval problem in rings containing a malicious host. In *Proceedings of the 17th International Conference on Structural Information and Communication Complexity* (2010), SIROCCO'10, Springer, pp. 157–167.
30. Luccio, F. L., and Markou, E. Mobile agents rendezvous in spite of a malicious agent. *arXiv preprint arXiv:1410.4772* (2014).
31. Peng, M., Shi, W., Corriveau, J.-P., Pazzi, R., and Wang, Y. Black hole search in computer networks: State-of-the-art, challenges and future directions. *Journal of Parallel and Distributed Computing 88* (2016), 1–15.
32. Shi, W., Garcia-Alfaro, J., and Corriveau, J.-P. Searching for a black hole in interconnected networks using mobile agents and tokens. *Journal of Parallel and Distributed Computing 74*, 1 (2014), 1945–1958.
33. Szor, P. *The art of computer virus research and defense*. Pearson Education, 2005.