Repairing Faulty Nodes and Locating a Dynamically Spawned Black Hole Search Using Tokens

Mengfei Peng
Aurora Technology Development
Toronto, Canada
Email: mengfei@auroratd.com

Wei Shi
School of Information Technology
Carleton University
Ottawa, Canada
Email: wei.shi@carleton.ca

Jean-Pierre Corriveau
School of Computer Science
Carleton University
Ottawa, Canada
Email: jeanpier@scs.carleton.ca

***Abstract*—In a distributed cloud, it is crucial to detect and eliminate faulty network entities in order to protect network assets and mitigate the risks associated with constantly arising attacks. 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. In this paper, we introduce a specific attack model that involves multiple faulty nodes that can be repaired by mobile software agents, as well as what we call a *gray virus* that can infect a previously repaired faulty node and turn it into a black hole. The *Faulty Node Repair and Dynamically Spawned Black Hole Search* (*FNR-DSBHS*) problem that proceeds from this model is much more complex and realistic than the traditional *Black Hole Search* problem. We first explain why existing algorithms addressing the latter do not work under this new attack model. We then distinguish between a *one-stop* gray virus that, after infecting a faulty node that has been repaired, can no longer travel to other nodes; and a *multi-stop* gray virus. We observe that, in an asynchronous network, a solution to the *FNR-DSBHS* problem is possible only when dealing with a single one-stop gray virus. In that specific context, we present a solution for an asynchronous ring network using a *token model*, that is, a ring in which a constant number of tokens is the *only* means of communication between the team of agents. We claim that, in such a ring, $b+9$ agents can repair all faulty nodes as well as locate *the* black hole that is infected by this single one-stop gray virus. We prove the correctness of the proposed solution and analyze its complexity in terms of number of mobile agents used and total number of moves performed by these agents. We show that in the worst case, within $O(kn^2)$ moves, $b+9$ agents suffice to repair b faulty nodes and report the location of the black hole that is infected, at any arbitrary point in time, by the one-stop gray virus.**

***Keywords*—Faulty Node Repair, Faulty Node, Dynamically Spawned Black Hole Search, Mobile Agent, Token, Asynchronous Network.**

I. 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 are usually geographically dispersed (as opposed to directly connected to each other), physically locating a network fault may be expensive and difficult, nay impossible. Consequently, using software agents to locate and/or repair network faults has become an alternative solution that has attracted considerable attention, especially in distributed computing [1]. A *mobile agent* is an abstract and autonomous software entity. As such, these agents are versatile and robust in changing environments, and can be programmed to work in cooperative teams. Such team members may have different complementary specialties, or be duplicates of one another [2].

There are many types of faults in a network, such as black holes (e.g., [3]), repairable black holes (e.g., [4], [5]), faulty agents (e.g., [6]), *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 [7]. Existing work has focused on the *Localization* and/or *Elimination* of network faults when the existence and the status of such faults are known a priori and do not change. Real world examples show that such models are no longer sufficient to cover new actual situations: nowadays many computer faults/viruses cannot be completely removed by anti-virus software. In particular, after a repair, a previously infected node may still be more vulnerable than a normal one and may be easily reinfected. For example, for the fast spreading worms mentioned in [8] (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*. In cloud computing, the term *vulnerability* refers to the flaws of a system that allow an attack to be successful [9]. Security issues pertaining to vulnerability have been widely discussed in research work (e.g., [10], [11], [12]). In the traditional black hole model, a hacker can inject into a computer host a virus that can delete any incoming data, this virus being possibly subsequently removed by an anti-virus agent. In the attack model that we consider in this paper, the initial fault of a node is repairable and we call each such node a *faulty node*. Once repaired, a faulty node behaves like a normal one; however, it can be infected again by what we call a *gray virus* (*GV* for brevity). The latter is a piece of malicious software that can infect a repaired node (due to this node's inherent vulnerability) by residing in it and turning it into a black hole. This GV has no effect on a normal node or link. A GV is called a *one-stop* gray virus if, after infecting a vulnerable node (i.e., a faulty node that has been repaired), it can no longer travel to other nodes; otherwise, it's a *multi-stop* gray virus. *When* a GV arrives in a host and *how long* it stays in it are unknown. Thus, a faulty node may be GV-infected only

momentarily or, in the worst case, permanently.

The attack model we have just sketched out leads to what we call the *Faulty Node Repair and Dynamically Spawned Black Hole Search* (*FNR-DSBHS*) problem. As with the traditional black hole search problem, this problem can be tackled using one or a team of identical mobile agents (hereafter agents). 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 one node to its neighbours. 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) [13]. As faulty nodes are harmful but can be repaired, part of these mobile agents' job is to repair all the faulty nodes. The agents then need to locate, within finite time, the GV(s) that infect some of the vulnerable nodes (by "turning" them into black holes). We remark that while an anti-virus software is scanning a computer host, this host is much harder to infect. Thus, similarly, we assume that a node cannot be turned into a black hole while visited by an agent.

There is typically a cost for repairing faults using software agents. For example, part of the content of an agent may be a repair kit that, once used, prevents this agent from further exploring the network [4]. In this research, we are interested in studying the worst case scenario namely: the cost of a repair is the worst (i.e., the most expensive in terms of number of agents used). To do so, we assume an agent "dies" after having repaired a faulty node. Moreover, should several agents simultaneously enter a faulty node, we postulate one agent will die after repairing the fault, whereas all other agents die immediately. Clearly, in the case of an agent that does not die after a repair and/or of other agents in the same faulty node that also survive after the repair, fewer agents will be needed to solve the problem.

Furthermore, contrary to the traditional black hole search in which all agents start in a network with one and only one back hole whose existence is known a priori, in our proposed new attack model, a repaired faulty node can be infected and turned into a black hole at any point in time while the agents traverse the network to try to repair the faulty nodes. This detail drastically changes the nature of the problem at hand in asynchronous networks: the possible scenarios in this case are significantly more complex than for the traditional black hole search, especially with the presence of multiple faulty nodes in need of repair, each of which eliminating all agents that simultaneously enter it.

Another difference between a faulty node and a black hole is that the former may eliminate multiple agents once and only once while the latter eliminates multiple agents continuously. The co-existence of multiple faulty nodes and black holes in our model significantly increases the difficulty of both repairing faulty nodes and locating the black hole because no agent knows the difference between the two types of nodes a priori.

Finally, we emphasize that in the field of black hole searching, network synchronization must be the first concern as it can lead to completely different tactics when solving the problem. In this paper, we specifically consider asynchronous networks. A *synchronous network* allows agents to traverse the network in globally timed steps. In each step, each agent can perform local computation, which includes exchanging information with other agents that are at the same processing step in the same node, and can then move to a neighbouring node (or remain in the same node) [4]. In such networks, a *time-out mechanism* has been introduced to enforce time synchronization (e.g., [14], [15]). Such a mechanism makes the black hole search problem easier. For example, in a network with a single black hole, let a team of two agents explore together, one as the leader and the second as the follower. After each time-out, the follower should know whether the leader has died in the black hole or not. Conversely, in asynchronous networks, there is no global clock mechanism. As such, the agents could initially wake up at different times. The time that an agent takes for every action (sleep or transit) is finite but unpredictable [16]. Therefore, it is impossible to distinguish whether an agent died in a black hole or is just stuck in a slow link/node in the network since the latter takes an unpredictable amount of time [17]. As a result, the black hole search is only solvable in an asynchronous network when the number of black holes is known a priori. In fact, Flocchini *et. al.* conclude that the black hole location can be known only after the *entire* network, except for the black holes, is explored [16]. That is, both the network size and the number of black holes must be known a priori for both single and multiple black holes search.

In the context of our new attack model, it is important to note that in an asynchronous network, a GV's moving speed is unpredictable. This leads to a crucial observation:

Observation 1: In an asynchronous network, given a multi-stop GV could move much faster than the agents, from an agent's viewpoint all the repaired nodes could appear to be black holes. Consequently, in the presence of a multi-stop GV, the *FNR-DSBHS* problem becomes unsolvable in an asynchronous network.

This is essentially the same conclusion as for the Multiple Black Hole Search problem: unless the connectivity of the network can be somehow guaranteed, multiple black holes may disconnect some regions of the network that, consequently, may never be visited by any agent. For the same reason, if a network contains more than 1 one-stop GV, some parts of the network may become disconnected when these GVs turn some nodes into black holes. That is, in an asynchronous network, in the presence of several one-stop GVs, the *FNR-DSBHS* problem also becomes unsolvable. Consequently, in this paper, we focus on solving the *FNR-DSBHS* problem with k faulty nodes and specifically 1 one-stop GV in an asynchronous network. We further restrict ourselves to a specific topology and to a specification communication model. Namely, here, we consider an un-oriented ring in which a constant number of tokens is the *only* means of communication between the team of agents.

II. Related Work

Dobrev *et al.* [7] study an asynchronous ring network with both co-located agents (initially located at the same node) and dispersed agents (initially located at different nodes). A *whiteboard model* is introduced in [7], i.e., each node has a bounded amount of storage for agents to communicate. Also

in ring networks, an *enhanced token model* has been used in [14]. A token is an atomic entity that the agents can see, place in the middle of a node or on a port, and/or remove [13]. Flocchini *et al.* [18] use a *pure token model* in which the tokens can be placed only in the middle of a node. The authors show that 2 co-located agents, each with 1 token, can locate the black hole in $\Theta(n \log n)$ moves not only in rings but also in an arbitrary network with a map.

Due to the characteristics of synchronous networks, using a face-to-face model associated with the time-out mechanism, a single black hole can be located in any network with only 2 co-located agents. In a *face-to-face model*, agents move through the network in synchronous steps and communicate only when they meet at a node [19]. The problem of finding the most efficient solution for the black hole search under the same assumption, 2 co-located agents searching for a black hole in an edge-labeled undirected synchronous network, has been considered in [20]. Klasing *et al.* [20] 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.

Unlike the case in the synchronous network, the black hole search problem in an asynchronous network is much more complex and more significant in practice. Dobrev *et al.* [21] introduce an algorithm to locate the black hole in an un-oriented ring network with dispersed agents in $O(kn+n \log n)$ moves. Shi *et al.* [22] prove that 2 co-located agents, each with $O(1)$ tokens, can locate the black hole in $\Theta(n)$ moves for hypercubes, tori and complete networks. Moreover, for an arbitrary unknown network graph with known n, Dobrev *et al.* [23] 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.

Cooper *et al.* [19] start studying the multiple black holes search (MBHS for brevity) problem in synchronous networks by locating all the reachable black holes. Later, the same authors [4] present solutions to the multiple repairable black holes (faulty nodes) problem. D'Emidio *et al.* [5] study the same problem under the same condition as [4] with the 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 [24]. 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.

Cai *et al.* [6] study a network decontamination problem with a black virus, which is related to both black hole search and intruder capture problems. The authors present a black virus that is a dangerous process that is initially resident in the network. Luccio *et al.* [25] consider a mobile agents rendezvous problem in spite of a malicious agent, which is similar to [26] that rendezvouses agents in a ring in spite of a black hole. Královič *et al.* [27] research a periodic data retrieval problem using a whiteboard in asynchronous ring networks with a malicious host. Bampas *et al.* [28] improve this result by showing that at least 4 agents are required when the malicious host is a gray hole that 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.

III. Model, Assumptions and Observations

Let $G = (E, V)$ be an un-oriented ring network, where E denotes the set of edges, V denotes the set of nodes (e.g., computer hosts) and n ($n = |V|$) denotes the number of nodes in G. Every pair $(u, v) \in E$ ($u \in V$ and $v \in V$) represents the link from neighbouring nodes u to v.

Let $V_f \subseteq V$ denote a set of faulty nodes, and b ($b < n$) denote the number of faulty nodes in the network. A faulty node deletes any incoming data. However, it can be repaired by the first visiting agent. After repair, a faulty node behaves like a normal node and is referred to as a *repaired node*. However, unlike a real normal node that is never a faulty node, a repaired node can be infected by a *gray virus* and consequently turned into a black hole. If one or more agents simultaneously enter a faulty node, one agent will die after repairing the fault, and all other agents will die immediately.

Let $\mathcal{A}$ denote a set of k ($k \geq 2$) identical agents in the network. We make no assumptions on the amount of time required by an agent's actions (e.g., computation, or movement) except that it is finite; thus, the agents are asynchronous. All agents initially wake up in the same node hb ($hb \in V$) that is referred to as their $homebase$ and is assumed to be safe (e.g., not a faulty node or black hole). All agents know the network topology a priori. Each agent is endowed with a limited number of tokens that can be put on or picked up from a port or the centre of a node. Any token on a node is visible to all agents on the same node. Since the agents are invisible to each other, thus, the only way to exchange information is through the use of tokens. All agents know n and b. Because, minimally one agent dies in a faulty node in order to repair it, at least $b + 1$ agents are required to allow one surviving agent to report the locations of the repaired nodes (potential GV infected nodes).

The links in the network obey a FIFO rule; that is, the agents do not overtake each other when travelling over the same link in the same direction. In an un-oriented ring, the agents are not able to agree on a common sense of direction [29] (e.g., the $Left$ direction to one agent might be the $Right$ to another agent).

This FIFO rule is necessary to guarantee minimal cost in terms of number of agents in solving the *FNR-DSBHS* problem in asynchronous networks regardless of choice of communication model (i.e., whiteboards or tokens). As we have mentioned earlier, a faulty node can only eliminate the first set of agents entering it simultaneously, while a black hole always eliminates all agents it encounters. While visiting each node, an agent is not able to distinguish between a faulty node and a black hole. This is because an agent either is eliminated by a faulty node or a black hole, or passes through a node that is either safe (i.e., not a faulty node nor a black hole) or repaired. Thus, the least expensive way to distinguish the black hole from the faulty nodes is to a) make sure an agent leaves a notice before it dies in the faulty node and b) make sure 2 agents walk into the black hole sequentially via the same port. Consequently a port that leads to a black hole will have 2 messages (e.g., 2 tokens) while a port that leads

to a faulty node will have only 1 message on it. Obviously, without the FIFO rule, all simultaneously arriving agents will die in a faulty node and a black hole. Consequently, the messages these agents left will be meaningless. Thus, in order to terminate the algorithm within finite time with finite number of agents we must rely on the FIFO rule.

Unlike Cooper *et al.* [4] who solve the multiple black hole problem in synchronous networks with the advantage of the time-out mechanism, in this paper we aim at solving the FNR-DSBHS problem in an asynchronous network where every agent's move or computation costs a finite but unpredictable amount of time. In [4], the goal of the agents is only to repair all the faulty nodes without reporting their locations, given the network map is supplied in advance. In this paper, we define the *FNR-DSBHS* problem as the following: use a team of mobile agents to repair all the faulty nodes in the entire network and have finally at least one surviving agent that knows the location of the black hole(s) infected by a gray virus.

Furthermore, Cooper *et al.* assume that if two or more agents enter a faulty node at the same time, only one dies for the repair while the others can continue exploration. We consider a more challenging model: all agents that simultaneously enter a faulty node die when this node is repaired.

Observation 2: If the one-stop GV appears after all faulty nodes have been repaired, the *FNR-DSBHS* problem becomes a faulty node repair problem and a single black hole search problem with all the possible locations of the black hole known a priori.

In this case, the *FNR-DSBHS* problem is even easier than a traditional single black hole search problem. In this research, we are only interested in studying the scenario in which a one-stop GV may appear and infect a repaired node at a time prior to having all faulty nodes repaired. Contrary to the traditional black hole search that has one black hole before the search starts, having a black hole that appears at a random time significantly increases the difficulty and complexity of the task. We will further explain this observation later.

A. Double Cautious Walk With Tokens

To locate a black hole in a traditional black hole search in an asynchronous network there is a commonly used technique called *cautious walk*: a first agent leaves a mark indicating danger (a token or whiteboard message) on its current node before it enters into the next node that may be the black hole. When a second agent sees this mark, it will not enter that next node. This technique is used to minimize the loss of mobile agents. In the proposed FNR-DSBHS problem, we introduce a new technique: *Double Cautious Walk With Tokens*:

An agent A (see Figure 1) marks a port in node u as dangerous by placing a token at this port before moving to the next node v. Once arriving at node v, the agent marks the entering port as dangerous by placing another token at the port and immediately returns to node u. Upon its return, this agent will pick up the first token in node u to show that this port is no longer dangerous and move again to node v. While arriving, the agent picks up the second token on the port through which this agent enters v. We call this port its $entering$ port and the other port its $exiting$ port.

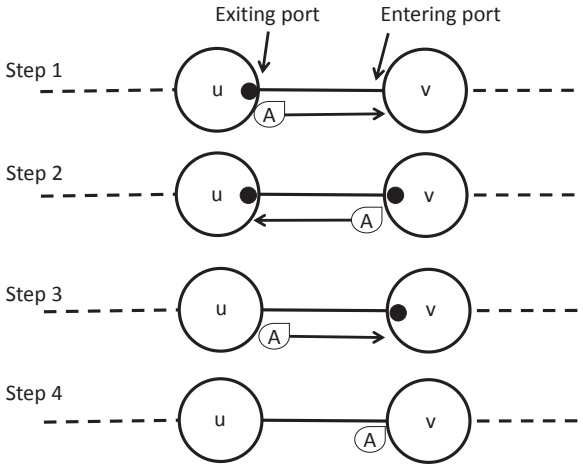

Fig. 1. Double Cautious Walk with tokens: Agent A puts a token at node u and moves to v (Step 1). Agent A puts a token at v and immediately returns to u (Step 2). Upon return, A picks up a token and again moves to v (Step 3). Upon arriving, A picks up a second token (Step 4).

Since the one-stop GV may appear and infect a vulnerable node at any point in time, a previously safe but vulnerable node u where an agent started the first step of a double cautious walk may be infected and become a black hole. This sudden change leads to the elimination of this agent while it returns to a previously safe node u during its double cautious walk (Step 2 in Figure 1). This is why the cautious walk with tokens [13] is no longer sufficient in terms of minimizing agent loss. In the new double cautious walk with tokens, even if an agent dies in the black hole while returning, it leaves a token at the second node, thus, further prevents extra agent loss.

During an exploration, some agents die after repairing faults, while some may die in a black hole. When an agent sees one token at a port, it knows that another agent is exploring the next node, but it cannot know whether the next node is a repaired node or a black hole. This agent needs to leave a second token at that port before entering the next node. If the next node is a black hole, the port that leads to the black hole will have two tokens on it since both agents died in it. Otherwise, if the next node becomes a repaired node after the previous agent dies, this agent will eventually pick up its first double cautious walk token. Thus, this mechanism distinguishes the black hole from a repaired node. An agent will never leave via a port with two tokens (*a 2-token port*) unless it is the same port through which it just entered a node.

The characteristic of the GV that can infect a repaired node at any time has significantly complicated the *FNR-DSBHS* problem. For a link (u, v), it is possible that 2 agents, A and B, enter via port $p1$ sequentially (not simultaneously due to the FIFO rule) after each putting down 1 token; this leaves 2 tokens on port $p1$ (see Figure 2). Both agents A and B proceed to node v and leave 2 tokens in total on $p2$.

In traditional black hole search, any node from which an agent comes is safe. Therefore, when the 2 agents return to node u via link (u, v), they know that nodes u and v are

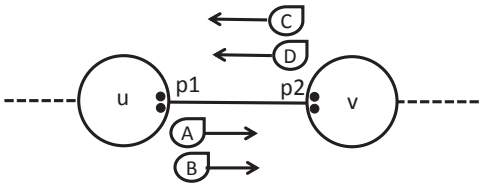

Fig. 2. During a double cautious walk, after visiting nodes u and v, two agents A and B die in a previously visited node u that is now a gray virus infected black hole.

safe. They will be able to safely return to their previous nodes regardless of the number of tokens left on the entering port. However, with a GV in the network, a previously visited node is no longer guaranteed to be safe. It is possible that the previously visited node u of agents A and B has been turned into a black hole. In this situation, both sets of 2 tokens on ports $p1$ and $p2$ will remain forever. This situation becomes more complex when there is one or more agent(s) traversing the ring in opposite directions (e.g., agents C and D in Figure 2).

It is very important to note that with multiple faulty nodes in the network as described in our new attack model, even when an agent can leave unlimited information on any node it visits, none of the existing black hole algorithms can solve the *FNR-DSBHS* problem. As previously mentioned, this is because 1) all traditional black hole search algorithms terminate when all nodes in the network have been explored except one and this sole unexplored one is the black hole; 2) in traditional black hole search algorithms, there is no mechanism to distinguish a black hole from faulty nodes; thus, all agents will treat a faulty node as a black hole. Since faulty nodes are all treated as black holes, agents will all be deleted by the faulty nodes. Thus, the "safe" area cannot grow to reach $n-1$ nodes (as required in the traditional black hole models) and no agent is able to explore $(n-1)$ nodes in a given network. Hence, even with extra communication capabilities and unlimited local memory storage on each node of the network, none of the existing black hole search algorithms can locate and repair all faulty nodes and locate the re-infected black hole.

B. List of Acronyms

Table I shows some acronyms used in this paper.

TABLE I. LIST OF ACRONYMS

Acronyms	Definition
k	Total Number of Agents
b	Total Number of Faulty Nodes
GV	Gray Virus
A, B, C, D, A_1, A_2	NameS of Agents
v, u, N_x, N_y	NameS of Nodes
hb	homebase Node
C_l, C_r	Number of tokens in the centre of an agent´s left/right block

In the next section, we present an algorithm that can repair all the faulty nodes, and locate the GV within finite time.

IV. ALGORITHM PAIR-BLOCK

A. General Description

An agent, A_1, wakes up at homebase hb and randomly chooses one direction to explore the ring using the double cautious walk. If A_1 is blocked by a 2-token port on node N_x, A_1 marks N_x as left-block in its memory. A_1 also memorizes whether there is any token in the centre of this left-block. A_1 reverses direction and continues exploration until blocked by another 2-token port on node N_y; A_1 remembers N_y as its right-block. An agent is said to be $Blocked$ when there are 2 or more tokens on the exiting port of a node in its current exploration direction. A_1 counts the distance between the left-block and right-block. If the distance is $n-2$, the algorithm terminates and the only unexplored node is the infected black hole; if the distance is smaller than $n-2$, A_1 executes the following:

If there is no token in the centre of either the left-block nor the right-block, A_1 puts 2 tokens in the centre of the right-block (2-token-centre node). A_1 then returns to look for its left-block until it arrives at its left-block or is blocked by another 2-token port. For the later case, the new node becomes the new left-block of A_1. A_1 puts 1 token in the centre of its left-block and waits. All other details of this procedure can be found in Procedure 2.

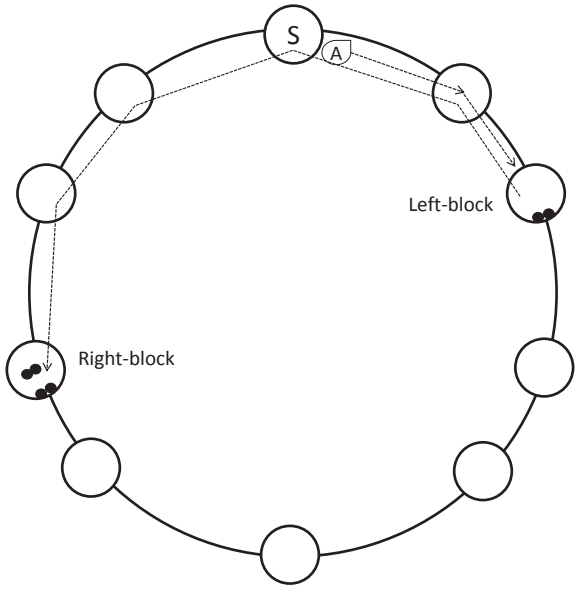

Fig. 3. The simplest situation: there are no other agents between the left-block and right-block of agent A

If the left-block is not beside a black hole, at least 1 token at the 2-token port will disappear. When A_1 sees 1 token is gone, it looks for its right-block to pick up the 2 tokens in the centre. A_1 then returns to its left-block and continues its exploration (see all situations in Procedure 4).

Another scenario could be that another agent A_2 picks up 1 token from the 2-token port of the right-block, if this right-block is not beside a black hole. When A_2 sees 2 tokens in the centre left by A_1, A_2 picks them up and becomes a $sender$. The sender goes to a 1-token-centre node (e.g., left-block of

TABLE II. NUMBER OF TOKENS ON NODES LEFT-BLOCK AND RIGHT-BLOCK. T: TOKEN. C: THE CENTRE.

Combi-nations	C_l	C_r	Action 1	Action 2
1	0	0	Put 2T in C of right-block. Go to left-block	Put 1T in C of left-block and wait.
2	0	1	Go to left-block. Put 2T in C of left-block	Go to right-block. Wait.
3	0	2	Go to left-block	Put 1T in C of left-block. Wait.
4	1	0	Put 2T in C of right-block	Go to left-block. Wait.
5	1	1	Put 1T in C of right-block	Go to left-block. Wait.
6	1	2	Go to left-block	Wait.
7	2	0	Put 1T in C of right-block	Exchange names. Wait.
8	2	1	Exchange names.	Wait.
9	2	2	Pick up 1T in C of right-block	Exchange names. Wait.

A_1), and drops the extra 2 tokens in the centre to inform A_1. After this, A_2 reverses its direction and continues its previous exploration. When A_1 sees 3 tokens in the centre, A_1 picks up all the tokens and goes to its right-block to continue exploration. Such exploration continues until one agent finds there are $n-1$ nodes between its left and right blocks.

The most complex scenario that costs the most agents is shown in Figure 4. According to algorithm Pair-Block, at most 2 agents may enter the same link sequentially from both sides. Consequently, at most 4 agents may move (2 in each direction) on the same link from both directions at the same time. If a node adjacent to this link becomes a black hole, all 4 agents on this link may die when they enter the black hole during their double cautious walk. The detailed proof is presented in Section V.

B. Detailed Explanation

a) Procedure "Pair-Block": An agent wakes up at hb and randomly chooses one port to explore the ring using the double cautious walk. When the agent returns to pick up its token during cautious walk, if it sees 2 tokens in the centre, it becomes a sender and sends the extra tokens to a 2-token port with 1 token in the centre. If an agent does not become a sender, it will meet a pair of left-block and right-block, and then either wait at its left-block, terminate the algorithm, or execute Procedure 2. In addition, *clear memory* means the agent deletes the information about left-block and right-block. After memory is cleared, an agent will memorize other nodes with 2-token ports as a new left-block or right-block.

b) Procedure "GoBack": To execute this procedure, an agent has met its left-block and is currently at its right-block. Let C_l and C_r denote the number of tokens in the centre of its left-block and right-block respectively. The agent puts i ($i = 0, 1$) token or picks up 1 token to make $C_l = 1 \& C_r = 2$. The agent then waits at its left-block. An agent may meet no/one/two tokens in the centre of its left-block and right-block. *Exchange names* means to exchange the names of its left-block and right-block; for example, nodes N_x and N_y are left and right blocks, respectively, so that after name exchange, N_y becomes left-block and N_x is right-block. All possibilities of C_l and C_r and actions 1 & 2 that an agent will take in these situations are shown in Table II.

Algorithm 1 PAIR-BLOCK

```
Wake up at hb. Choose one port to explore the ring using double cautious walk.
loop
    if Blocked by a port that has more than 2 tokens during exploration then
        Execute the same as being blocked by a 2-token port.
    end if
    if See 2 tokens in the centre when returning during double cautious walk then
        Pick up the 2 extra tokens in the centre. Become a sender.
        if Blocked by a 2-token port with one token in the centre then
            Put 2 tokens in the centre. Reverses direction. Clear memory.
        else if Blocked by a 2-token port then
            Reverse direction. Delete the 2 extra tokens. Clear memory.
        end if
    end if
    Continue exploration until blocked by a 2-token port. Mark this node left-block.
    Reverse direction and continue exploration until blocked by another 2-token port. Mark this node right-block.
    Count the distance d of left-block and right-block.
    if d = 0 then
        Wait until any 2-token port has fewer than 2 tokens. Leave through that port and continue exploration. Clear memory.
    else if 0 < d < n − 2 then
        Execute GoBack.
    else if d = n − 2 then
        Terminate the algorithm.
    end if
end loop
```

c) Procedure "GoToOne": The agent is currently at its right-block and needs to move to its left-block. It may arrive at its left-block successfully or meet another 2-token port. There are three main situations that may happen and should be discussed whenever an agent moves from its left-block to its right-block .

1) A_1 arrives at its left-block, and the 2-token port still has 2 tokens.
2) A_1 arrives at its left-block, and the 2-token port has fewer than 2 tokens.
3) A_1 is blocked by a 2-token port with no/one/two tokens in the centre before arriving at its left-block.

d) Procedure "WaitToFind": An agent has met its right-block and now waits at its left-block. The agent waits until 1 token on the exiting port is gone or $C_l = 3$. For the former case, the agent looks for the 2 tokens in the centre of its right-block. For the later case, the agent picks up all 3 tokens and goes to its right-block to continue exploration.

V. CORRECTNESS AND COMPLEXITY ANALYSIS

Lemma 1: $b+2$ agents are necessary to repair all faulty nodes and locate the black hole in a ring network

Proof: Since there are b faulty nodes in the ring network and 1 agent can only repair 1 faulty node, b agents are needed.

Algorithm 2 GoBack

if $C_l + C_r = 0$ **then**
  Put 2 tokens in the centre of right-block. $i = 1$. Execute **GoToOne**.
else if $C_l = 0 \& C_r = 1$ **then**
  Go to left-block
  if Arrive left-block and the 2-token port still has 2 tokens **then**
    Put 2 tokens in the centre of left-block. Exchange names. $i = 0$. Execute **GoToOne**.
  else if Arrive left-block and the 2-token port has fewer than 2 tokens **then**
    Clear memory.
  else if Blocked by a 2-token port before left-block **then**
    Update this node as left-block. Exchange names. Execute **GoBack**.
  end if
else if $C_l = 0 \& C_r = 2$ **then**
  $i = 1$. Execute **GoToOne**.
else if $C_l = 1 \& C_r = 0$ **then**
  Put 2 tokens in the centre of right-block. $i = 0$. Execute **GoToOne**.
else if $C_l = 1 \& C_r = 1$ **then**
  Put 1 token in the centre of right-block. $i = 0$. Execute **GoToOne**.
else if $C_l = 1 \& C_r = 2$ **then**
  $i = 0$. Execute **GoToOne**.
else if $C_l = 2 \& C_r = 0$ **then**
  Put 1 token in the centre of right-block. Exchange names. Execute **WaitToFind**.
else if $C_l = 2 \& C_r = 1$ **then**
  Exchange names. Execute **WaitToFind**.
else if $C_l = 2 \& C_r = 2$ **then**
  Pick up 1 token in the centre of right-block. Exchange names. Execute **WaitToFind**.
end if

Algorithm 3 GoToOne

if Arrive left-block and the 2-token port still has 2 tokens **then**
  Put i token in the centre of left-block. Execute **WaitToFind**.
else if Arrive left-block, and the 2-token port has fewer than 2 tokens **then**
  Go to right-block.
  if Arrive at right-block and see two tokens in the centre **then**
    Pick up all tokens in the centre. Reverse direction. Clear memory.
  else if Arrive at right-block **then**
    Reverse direction. Clear memory.
  else if Blocked by a 2-token port with 2 tokens in the centre **then**
    Pick up all tokens in the centre. Reverse direction. Clear memory.
  else if Blocked by a 2-token port **then**
    Reverse direction. Clear memory.
  end if
else if Blocked by a 2-token port with no tokens in the centre **then**
  Put 1 token in the centre, update this node as left-block. Execute **WaitToFind**.
else if Blocked by a 2-token port with 1 token in the centre **then**
  Update this node as left-block. Execute **WaitToFind**.
else if Blocked by a 2-token port with 2 tokens in the centre **then**
  Pick up 1 token. Update this node as left-block. Execute **WaitToFind**.
end if

Algorithm 4 WaitToFind

Wait till either 1 token on the exiting port is gone or 1 token in the centre becomes 3.
if 1 token on the exiting port is gone. **then**
  Go to right-block.
  if Blocked by a 2-token port with 2 tokens in the centre **then**
    Pick up 2 tokens in the centre. Return to left-block.
  else if Blocked by a 2-token port or arrive right-block. **then**
    Return to left-block.
  end if
  if Arrive at left-block and there is 1 tokens in the centre **then**
    Pick up the token in the centre. Clear memory.
  else if Blocked by a 2-token port. **then**
    Reverse direction. Clear memory.
  else
    Clear memory.
  end if
else if 1 token in the centre becomes 3 **then**
  Pick up all 3 tokens in the centre. Go to right-block and clear memory.
end if

To distinguish the black hole from the repaired nodes, at least one agent has to enter the black hole and die, thus, $b+1$ agents are required. To report the locations of the faulty nodes and the black hole, at least 1 agent has to survive, hence, $b + 2$ agents are necessary. ■

Lemma 2: $b+9$ agents are sufficient to report all repaired faulty nodes and locate the black hole in a ring network.

Proof: As proven in Lemma 1, b agents are needed to repair all faulty nodes. In our algorithm, to distinguish the black hole from the repaired nodes, each port that leads to the black hole will have 2 tokens, each of which was left by one agent. Because there is already 1 token left from the previous faulty node repair, 3 more tokens from three different agents will be needed. At least 1 agent has to survive and report the locations, thus, minimally $b + 4$ agents are required to repair all the faulty nodes and locate the black hole that may appear at any arbitrary point in time.

If two agents simultaneously enter the same faulty node, they both die. Since the agents travelling on the same link in the same direction obey the FIFO rule, maximum 2 agents (one from each direction) can enter the same faulty node and simultaneously die. If 2 agents meet from different directions, all faults from their common homebase hb to the meeting node have been repaired. Therefore, in this ring network, there is only one black hole and no faulty nodes after this meeting, so that one more agent is needed in this situation and $b + 5$ agents are necessary.

As a GV can infect a repaired node at any time, it is also possible that node v turns into a black hole while an agent is returning to v after picking up its first token (Step 3 in Figure 1), which makes the agent die in the black hole without leaving any trace in the network. For a repaired node, 1 agent has already died repairing it and left 1 token on a port that leads to this repaired node, so that 3 more agents can enter node v and die.

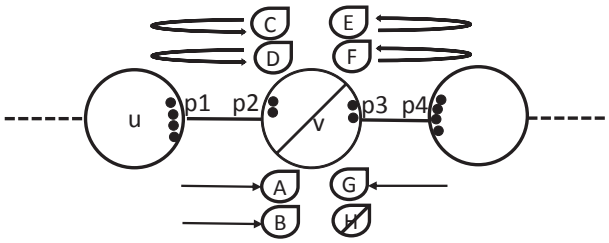

Fig. 4. At most 8 agents die in the black hole

Assume agents A and B travel in the opposite direction as agents C and D. They left their first cautious walk tokens on port p1 and p2, respectively. While agents A and B are moving towards node v, node v becomes infected and turned into a black hole, so that agents A and B will immediately die when they arrive at node v. Agents C and D will also die in node v after they leave their second cautious walk tokens on port p1. None of these 4 agents can return to pick up their tokens on port p1 (see Figure 4). According to Line 4 in Procedure 1, all agents treat such a port the same as a 2-token port.

If node v can be turned into a black hole, it is a repaired node, so that there has been an agent H who died for repairing it. Since agent H immediately dies after arriving at a faulty node, it cannot leave its second double cautious walk token, thus, it leaves only 1 token on port p4. If agents G, E, and F die in the same situation as agents A, C, and D, there will be 2 and 4 tokens on port p3 and p4 respectively. In summary, 7 agents die in the black hole and 1 agent dies repairing a faulty node. This is true even if agents A, B, and G die while returning after picking up their first token instead of moving towards node v, because the second cautious walk tokens left on port p1 by agents C and D, as well as the 2 tokens on port p4 by agents E and F, will remain and block other agents. Since in the worst case only agents A, B, G, and H die in the black hole, 4 extra agents die in this situation, which requires more agents than the above situation where only 3 agents die without leaving any trace. These two situations cannot happen in the same network. Therefore, $b + 9$ agents are required in total.

■

Lemma 3: Algorithm Pair-Block can correctly repair all faulty nodes and locate the black hole.

Proof: For any port with 2 or more tokens, there are three cases to consider for the agents that left these tokens: all active agents, only one dead agent (for fault repair), at least two dead agents. For the first two cases, all active agents will return to pick up their tokens and leave at most 1 token on the port, so that the port can only temporarily block other agents. Since all links obey the FIFO rule, a port with 2 or more dead agents leads to a black hole. An agent will be permanently blocked by such a port in one direction, hence, it can only explore the ring in the opposite direction and either die or arrive at the other side of the black hole. If an agent has been blocked by 2 ports that lead to the black hole, it has visited $n - 1$ nodes and located the black hole. ■

Lemma 4: In the worst case, the *FNR-DSBHS* task can finish within $O(kn^2)$ moves in total.

Proof: As proved in Lemma 2, in the worst case all accidents happen and leave only 1 surviving agent. As there is a single black hole in the network and each port that leads to the black hole will have at least 2 tokens, the surviving agent will be blocked by the two ports and the distance is $n - 2$, thus, the algorithm is terminated.

For any agent, one step of the double cautious walk approach needs 3 actual moves. According to Procedure 1 lines 14 and 15, forming a pair of left-block and right-block needs up to $2 * 3(n - 2)$ moves. Also in Procedure 1 or Procedure 4, removing a pair may also need up to $2*3(n-2)$ moves. If every exploration of a new node by an agent creates a pair of left and right blocks, up to $(2*3+2*3)*(n-2)*n$ moves are required. Furthermore, if this happens to every agent, a total of $12*k*n*(n-2)$ moves are required to form and remove a pair of blocks for all agents. Hence, $O(kn^2)$ moves are required to repair all faulty nodes and locate the black hole in the worst case. ■

Theorem 1: After $O(kn^2)$ moves, $b + 9$ agents are sufficient to repair and locate b faulty nodes and the black hole infected by a one-stop gray virus at a random time.

VI. Conclusion and Future Work

In this paper, we present a new attack model involving both faulty nodes and a so-called gray virus (that can only infect a faulty node that was previously repaired). Most importantly, *at any point in time*, this virus may turn into a black hole any node previously visited by an agent, as well as unvisited nodes and even a node being currently visited by an agent. Such dynamic behaviour leads to what we call the *Faulty Node Repair and Dynamically Spawned Black Hole Search* (*FNR-DSBHS*) problem. The latter is considerably more realistic and complex than the traditional black hole search problem (in which black holes are not dynamic but instead exist from the very start of the algorithm trying to locate them). We consider a specific instance of this problem and introduce an algorithm that can repair all faulty nodes and locate the black hole that is infected by a one-stop GV in an asynchronous ring network using a *token model*, that is, a ring in which a constant number of tokens is the *only* means of communication between the team of agents. Our solution rests on a new technique called *double cautious walk with tokens*, which can mark both the previously visited and unexplored dangerous nodes. We demonstrate this technique guarantees minimal loss of mobile agents. We conclude that $b+9$ agents are required to repair all faulty nodes as well as locate the black hole that is infected by the one-stop gray virus.

We emphasize that, without any additional assumptions, the general *FNR-DSBHS* problem becomes a multiple black hole search problem and remains unsolvable in an asynchronous network. In particular, as future work, we ask what additional assumptions must be made in order for this problem to become solvable in the presence of several one-stop GVs and in the case of one or more multi-stop GVs in an arbitrary unknown network topology. For example, in order to make the detection of a multi-stop gray virus more feasible, we could decree that such a GV can only move to another node

after deleting at least one agent at the current node (instead of freely moving at any point in time, which is the general case). Also, we must eventually ask whether there are fundamental differences between one-stop GVs and multi-stop ones if GVs are truly dynamic, that is, if GVs may appear in the network at any point in time. We may also consider whether the general problem or any of its specializations is solvable in a synchronous network.

Additionally, for a one-stop GV in an asynchronous network, apart from ring networks, it will be necessary to consider other common network topologies as well as arbitrary networks. Also, in order to reduce the number of agents required, is it possible to avoid multiple simultaneous agent deaths in a same node? Finally, beyond the token model, we may want to consider other communication models such as face-to-face or whiteboards.

ACKNOWLEDGMENTS

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

REFERENCES

[1] R. Královič, "Advice complexity: Quantitative approach to a-priori information," in *SOFSEM 2014: Theory and Practice of Computer Science*. Springer, 2014, pp. 21–29.

[2] M. S. Greenberg, J. C. Byington, and D. G. Harper, "Mobile agents and security," *Communications Magazine, IEEE*, vol. 36, no. 7, pp. 76–85, 1998.

[3] S. Dobrev, P. Flocchini, R. Královič, and N. Santoro, "Exploring an unknown dangerous graph using tokens," *Theoretical Computer Science*, vol. 472, pp. 28–45, 2013.

[4] C. Cooper, R. Klasing, and T. Radzik, "Locating and repairing faults in a network with mobile agents," *Theoretical Computer Science*, vol. 411, no. 14-15, pp. 1638–1647, Mar. 2010.

[5] M. D'Emidio, D. Frigioni, and A. Navarra, "Exploring and making safe dangerous networks using mobile entities," in *Ad-hoc, Mobile, and Wireless Network*. Springer, 2013, pp. 136–147.

[6] J. Cai, P. Flocchini, and N. Santoro, "Network decontamination from a black virus," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 2013, pp. 696–705.

[7] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro, "Mobile search for a black hole in an anonymous ring," in *Proceedings of the 15th International Conference on Distributed Computing*, ser. DISC '01. London, UK, UK: Springer-Verlag, 2001, pp. 166–179.

[8] P. Szor, *The art of computer virus research and defense*. Pearson Education, 2005.

[9] K. Hashizume, D. G. Rosado, E. Fernández-Medina, and E. B. Fernandez, "An analysis of security issues for cloud computing," *Journal of Internet Services and Applications*, vol. 4, no. 1, pp. 1–13, 2013.

[10] M. Almorsy, J. Grundy, and I. Müller, "An analysis of the cloud computing security problem," in *Proceedings of APSEC 2010 Cloud Workshop, Sydney, Australia, 30th Nov*, 2010.

[11] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009, pp. 85–90.

[12] B. Grobauer, T. Walloschek, and E. Stocker, "Understanding cloud computing vulnerabilities," *Security & privacy, IEEE*, vol. 9, no. 2, pp. 50–57, 2011.

[13] S. Dobrev, R. Královič, N. Santoro, and W. Shi, "Black hole search in asynchronous rings using tokens," in *The 6th Conference on Algorithms and Complexity (CIAC '06)*. Springer, 2006, pp. 139–150.

[14] J. Chalopin, S. Das, A. Labourel, and E. Markou, "Tight bounds for black hole search with scattered agents in synchronous rings," *Theoretical Computer Science*, 2013.

[15] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc, "Searching for a black hole in synchronous tree networks," *Combinatorics, Probability & Computing*, vol. 16, no. 4, pp. 595–619, Jul. 2007.

[16] P. Flocchini and N. Santoro, "Distributed security algorithms by mobile agents," in *Distributed Computing and Networking*. Springer, 2006, pp. 1–14.

[17] W. Shi, "Black hole search with tokens in interconnected networks," in *The 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2009)*. Springer, 2009, pp. 670–682.

[18] P. Flocchini, D. Ilcinkas, and N. Santoro, "Ping pong in dangerous graphs: Optimal black hole search with pure tokens," in *Proceedings of the 22nd International Symposium on Distributed Computing*. Springer, 2008, pp. 227–241.

[19] C. Cooper, R. Klasing, and T. Radzik, "Searching for black-hole faults in a network using multiple agents," in *Proceedings of the 10th International Conference on Principles of Distributed Systems*, ser. OPODIS'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 320–332.

[20] R. Klasing, E. Markou, T. Radzik, and F. Sarracco, "Approximation bounds for black hole search problems," *Networks*, vol. 52, no. 4, pp. 216–226, 2008.

[21] S. Dobrev, N. Santoro, and W. Shi, "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)*. Springer, 2007, pp. 608–617.

[22] W. Shi, J. Garcia-Alfaro, and J.-P. Corriveau, "Searching for a black hole in interconnected networks using mobile agents and tokens," *Journal of Parallel and Distributed Computing*, vol. 74, no. 1, pp. 1945–1958, 2014.

[23] S. Dobrev, P. Flocchini, R. Královič, and N. Santoro, "Exploring an unknown graph to locate a black hole using tokens," in *Fourth IFIP International Conference on Theoretical Computer Science-TCS 2006*. Springer, 2006, pp. 131–150.

[24] P. Flocchini, M. Kellett, P. C. Mason, and N. Santoro, "Mapping an unfriendly subway system," in *Proceedings of the 5th International Conference on Fun with Algorithms*. Springer, 2010, pp. 190–201.

[25] F. L. Luccio and E. Markou, "Mobile agents rendezvous in spite of a malicious agent," *arXiv preprint arXiv:1410.4772*, 2014.

[26] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro, "Multiple agents rendezvous in a ring in spite of a black hole," in *Proceedings of the 7th International Conference on Principles of Distributed Systems*. Springer, 2004, pp. 34–46.

[27] R. Královič and S. Miklík, "Periodic data retrieval problem in rings containing a malicious host," in *Proceedings of the 17th International Conference on Structural Information and Communication Complexity*, ser. SIROCCO'10. Springer, 2010, pp. 157–167.

[28] E. Bampas, N. Leonardos, E. Markou, A. Pagourtzis, and M. Petrolia, "Improved periodic data retrieval in asynchronous rings with a faulty host," in *Structural Information and Communication Complexity*. Springer, 2014, pp. 355–370.

[29] S. Dobrev, N. Santoro, and W. Shi, "Using scattered mobile agents to locate a black hole in an un-oriented ring with tokens," *International Journal of Foundations of Computer Science*, vol. 19, no. 06, pp. 1355–1372, 2008.