Black Hole Search in Asynchronous Rings Using Tokens

S. Dobrev[1], R. Královič[2], N. Santoro[3], and W. Shi[3]

[1] School of Information Technology and Engineering, University of Ottawa, Ottawa, K1N 6N5, Canada
[2] Dept. of Computer Science, Comenius University, Mlynska dolina, 84248 Bratislava, Slovakia
[3] School of Computer Science, Carleton University, Ottawa, K1S 5B6, Canada

Abstract. A *black hole* is a highly harmful host that disposes of visiting agents upon their arrival. It is known that it is possible for a team of mobile agents to locate a black hole in an asynchronous *ring* network if each node is equipped with a *whiteboard* of at least $O(\log n)$ dedicated bits of storage. In this paper, we consider the less powerful *token model*: each agent has has available a bounded number of tokens that can be carried, placed on a node or removed from it. All tokens are identical (i.e., indistinguishable) and no other form of communication or coordination is available to the agents. We first of all prove that a team of two agents is sufficient to locate the black hole in finite time even in this weaker coordination model. Furthermore, we prove that this can be accomplished using only $O(n \log n)$ moves in total, which is optimal, the same as with whiteboards. Finally, we show that to achieve this result the agents need to use only $O(1)$ tokens each.

1 Introduction

1.1 The Framework

Whereas exploration problems by mobile agents have been extensively studied in the context of *safe* networks, the reality of networked systems supporting mobility agents is that these systems are highly *unsafe*. Indeed, the most pressing concerns are all about security issues and mainly in regards to the presence of a *harmful host* (i.e., a network node damaging incoming agents) or of a *harmful agent* (e.g., a mobile virus infecting the network nodes) [2, 10, 11, 13, 14].

The computational and algorithmic research has just recently started to consider these issues. The computational issues related to the presence of a harmful agent have been explored in the context of intruder capture and network decontamination; in the case of harmful host the focus has been on the *black hole* (BH), a node that disposes of any incoming agent without leaving any observable trace of this destruction [3, 5–7, 12]. In this paper, we continue the investigation of the black hole search problem.

As mentioned, a *black hole* is a network site that disposes of any incoming agent without leaving any observable trace of this destruction. It models e.g.

a node where a resident process (e.g., an unknowingly installed virus) deletes visiting agents or incoming data; furthermore, any undetectable crash failure of a site in an asynchronous network transforms that site into a black hole. In presence of a black hole, the first important goal is to determine its location. To this end, a team of mobile system agents is deployed; their task is completed if, within finite time, at least one agent survives and knows the links leading to the black hole. The research concern is to determine under what conditions and at what cost mobile agents can successfully accomplish this task, called the *black hole search* (BHS) problem. The main complexity parameter is the *size* of the team; i.e., the number of agents used in the search. Another important measure is the amount of *moves* performed by the agents in their search.

Both solvability and complexity of BHS depend on a variety of factors, first and foremost on whether the system is *asynchronous* [4–7] or *synchronous* [3, 12]. Indeed the nature of the problem changes drastically and dramatically. For example, both in synchronous and asynchronous systems, with enough agents it is possible to locate the black hole if we are aware of its existence; however, if there is doubt on whether or not there is a black hole in the system, in absence of synchrony this doubt can *not* be removed. In fact, in an asynchronous system, it is *undecidable* to determine if there is a black hole [6]. The consequences of this fact are numerous and render the asynchronous case considerably difficult. In this paper we continue the investigation of the asynchronous case.

Other important factors influencing solvability and complexity are the amount of a priori knowledge held by the agents (e.g., number n of nodes, map of network, etc.) and the means offered by the system for agents communication and coordination (e.g., whiteboard, blackboard, reliable message passing, etc). In particular, all existing investigations on BHS in asynchronous systems have assumed the presence of a powerful inter-agent communication mechanism, *whiteboards*, at all nodes. In the whiteboard model, each node has available a local storage area (the whiteboard) accessible in fair mutual exclusion to all incoming agents; upon gaining access, the agent can write messages on the whiteboard and can read all previously written messages. This mechanism can be used by the agents to communicate and mark nodes or/and edges, and has been commonly employed in several mobile agents computing investigations (e.g. see [1, 8, 9]).

Although many research questions are still open, the existing investigations have provided a strong characterization of the asynchronous BHS problem using whiteboard. In particular, it is possible for two agents with a map of the network to determine the location of the black hole in any bi-connected graph[4] using $O(n \log n)$ moves, provided there are $O(\log^2 n)$ bits whiteboards [6].

In this paper we consider an asynchronous *ring* network. The ring is the sparsest bi-connected graph and the one for which the cost (in terms of number of moves) for black hole search with whiteboards is the worst: $\Omega(n \log n)$. Using quite a different protocol, two agents can however locate the black hole with $O(n \log n)$ moves using $O(\log n)$ bits whiteboards [5].

[4] edge bi-connectivity is required for BHS in asynchronous systems [6]

We consider ring networks in the less powerful *token* model, often employed in the exploration of safe graphs. In this model, each agent has available a bounded number of tokens that can be carried, placed on a node or on a port or removed from it. All tokens are identical (i.e., indistinguishable) and no other form of communication or coordination is available. Some natural questions immediately arise: is the BHS problem is still solvable with this weaker mechanism, and if so under what conditions and at what cost. Notice that the use of tokens introduces another complexity measure: the number of tokens. Indeed, if the number of tokens is unbounded, it is possible to simulate a whiteboard environment; hence the question immediately arises of how many tokens are really needed.

1.2 Our Results

The network under consideration is an asynchronous ring network with a black hole. In such a network, in presence of whiteboards, the black hole search problem can be solved with team of just *two* agents, and performing only $\Theta(n \log n)$ moves [5]. We consider the same topology and examine the BHS problem using tokens.

We first of all prove that a team of *two* agents is sufficient to locate the black hole in finite time even in this weaker coordination model. Furthermore, we prove that this can be accomplished using only $O(n \log n)$ moves in total, which is optimal, the same as with whiteboards. Finally, we show that the agents need to use only $O(1)$ tokens. These results are established constructively: we do present protocols that allow a team of two agents to correctly locate the BH with that number of moves and with those few tokens. The first protocol uses a total of ten tokens, while in the second that number is reduced to 3.

Hence we show that, although tokens are a weaker means of communication and coordination, their use does not negatively affect solvability and it does not even lead to a degradation of performance. On the contrary, whereas the protocols using whiteboards assumed at least $O(\log n)$ dedicated bits of storage at each node, the ones proposed here use only three tokens in total.

2 Model and Basic Tools

2.1 The Model and Basic Observations

Let $\mathcal{R}$ be a anonymous ring of n nodes (i.e. all the nodes look the same, they do not have distinct identifiers). Operating on $\mathcal{R}$ is a set of k agents $a_1, a_2, ..., a_k$. The agents are *anonymous* (do not have distinct identifiers), *mobile* (can move from a node to a neighbouring node) and *autonomous*; each has computing and limited memory capabilities ($O(\log n)$ bits suffice for all our algorithms). All agents have the same behaviour, i.e. follow the same protocol, and start at the same node (however, they may start at different and unpredictable times), called *homebase* (HB for brevity). The agents do not know k, nor do they know how many agents have been awaken before. Since all agents start at the same node,

we can assume that the ring is *oriented*, i.e. all ports are labelled left and right consistently in the whole ring[5].

The agents can interact with their environment and with each other only through the means of *tokens*. A token is an atomic entity that the agents can see, place it in the middle of a node or on a port and/or remove it. Several tokens can be placed on the same place. The agents can detect the multiplicity, but the tokens themselves are undistinguishable from each other. Initially, there are no tokens placed in the network, and each agent starts with some fixed number of tokens (depending on the algorithm).

The basic computational step of an agent (executed either when the agent arrives to a node, or upon wake-up) is to *examine* the node (returns a triple of non-negative integers - multiplicity of tokens at the middle of the node, on the right port and on the left port, respectively), *modify* the tokens (by placing/removing some of the tokens at the current node) and either *fall asleep* or *leave* the node through either left or right port. The whole computational step is performed as an atomic action, i.e. as if it took no time to execute it.

The computation is asynchronous in the sense that the time an agent sleeps or is on transit is finite but unpredictable. The links obey FIFO rule, i.e. the agents do not overtake each other when traveling over the same link in the same direction.

Note that the tokens are the only means of inter-agent communication we consider. There is no read/write memory (whiteboards) in the nodes the agents can access, nor is there face-to-face communication. In fact, the agents do not even need to be capable of seeing each other - they only see the tokens.

One of the nodes of the ring $\mathcal{R}$ is highly harmful – it disposes of every agent that enters it, without leaving any trace of this destruction observable from the outside. Due to this behaviour, this node is called *Black Hole* (or BH for brevity). All the agents are aware of the presence of the BH, but at the beginning the location of the BH is unknown. The goal is to locate the BH, i.e. at the end there must be at least one agent that has not entered the BH and knows the location of the BH.

The primary complexity measure is *size*: the number of agents needed to locate the BH. Other complexity measures we are interested in are *token size*: the number of tokens each agent starts with, *cost*: the total number of moves executed by the agents (worst case over all possible timings) and *time*: the time it takes to locate the black hole, from the moment the *second*[6] agent wakes-up until the black hole is located, assuming transiting an edge takes at most one time step.

Since the first move of an agent can end up in the BH, we immediately get:

[5] the agents can simply use the notions of left/right according to the port labelling at the HB, remembering as they move how does the labelling at the current node relate to the labelling at the HB

[6] the first agent might immediately enter the black hole, and the second agent might wake-up arbitrarily late, resulting in unbounded time complexity if we measure from the time the first agent wakes up

Lemma 1. *[5] At least two agents are needed to locate the black hole.*

Because of the asynchrony, the agents can not distinguish a slow node from the Bh. From this we get:

Lemma 2. *[5] It is impossible to find the Black Hole if the size of the ring is not known.*

2.2 Basic Tools and Techniques

Cautious Walk with Token At any moment of execution, the ports can be classified as *unexplored* – no agent has exited or arrived via this port yet, *dangerous* – an agent has exited via this port, but no agent has arrived yet via it, or *safe* – an agent has arrived via this port.

As our primary complexity measure is the number of agents needed, we aim to prevent unnecessary agent disappearances by making sure that no two agents enter the Bh over the same link. This is achieved by forbidding the agents to leave a node via a dangerous port. This means the agents have to mark the dangerous ports. In order to facilitate progress, the agents are also required to remove the dangerous marks whenever they learn that a port marked dangerous in fact leads to a non-Bh node: When an agent that left dangerous mark on a port of node u arrives at node v, it immediately returns to u, and removes the dangerous mark to signal that the link from u to v has became *safe*. Afterwards (provided it is not interrupted by a message in u) the agent returns to v and proceeds from there.

This technique has been introduced in [5] and named *Cautious Walk*. In [5] the whiteboards have been used to mark dangerous and safe ports. In this paper, the tokens placed on a port (one or two, depending on context) mean the port is dangerous (however, there will be cases where tokens might appear also on non-dangerous ports); by default, there are no tokens on safe and unexplored ports and the agents distinguish them from the context. (As we aim to use $O(1)$ tokens overall, while $O(n)$ ports will eventually become safe, we cannot afford to mark the safe ports by tokens.)

Elimination Technique In order to minimize the *cost*, i.e. the total number of moves executed by all agents, we use elimination technique to effectively reduce the number of active agents to two – the first two agents to wake-up. The idea is to have the first two agents mark the Hb by their tokens, the agents waking up later check the marks at the Hb and become passive if they see there are already two agents active.

3 Algorithm *Divide with Token*

3.1 General Description and Ideas

A node is called *explored* if it has been visited by an agent, *unexplored* otherwise. The set of the explored nodes is called *explored region*; the part of the ring

consisting of unexplored nodes the *unexplored region*. As all agents start at the HB, the explored region (and thus also the unexplored region) is connected. The two extremal explored nodes having an unexplored neighbour are called Last-Safe-Place (LSP) (at the very beginning, the HB is the sole LSP). The LSPs keep changing while the explored region is getting larger and larger.

The main technique for locating the BH is borrowed from [5]: The two agents logically partition the unexplored region into two connected parts of (almost) equal size, and then each agent goes to explore its part using Cautious Walk. Since there is only one BH, exactly one agent (say agent a) will finish exploring its part. The agent a then traverses the explored part until it reaches the LSP of agent b. At this moment, a knows the distance between LSPs, i.e. the size of the explored part. Since n is known, a also knows the size of the unexplored part. If the unexplored part consists of a single node v, a determines that the BH is located at v and terminates the algorithm (b has already been terminated by entering the BH). Otherwise, a logically divides the remaining unexplored region into two connected, almost-equally sized work assignments W_a and W_b for a and b, such that W_b contains the node to which b is currently heading. Afterwards, a leaves a message for b informing it about W_b and goes on to explore W_a. The process is repeated until the unexplored region contains single node – the BH.

The above general description omits details of how the agents communicate – the format of the message, identifying the LSP, implementing Cautious Walk. In [5] whiteboards were used to store the message (and the status of the ports). In this paper, we aim to implement the above approach using only tokens, and in fact using three tokens in total.

The first step in this direction is to use tokens only for dangerous ports, not for safe or unexplored ones. This can be easily achieved, as all ports between the dangerous ports leaving from LSPs are implicitly safe. The second step – using tokens to encode the message is trickier. Note that it is sufficient to leave as a message for b not the size of W_b, but the number of times a has finished its part before b has made any progress. b can re-compute the size of its part by that many times halving the size of its last work assignment (and being careful to use the same rounding as a used when the unexplored area had odd size). This means the message needs only to be of size at most $\log n$ and allows a to update its message by simply incrementing it[7].

The basic technique we use is to encode the message by the distance between the LSP of b and a token (*end-of-message* marker) left by a in the explored area. In fact, in order to minimize the number of tokens used, a leaves a message x by moving b's cautious walk token x nodes in the direction opposite to b's exploring. This way, when/if b returns to move its token and does not find it there, it knows there is a message waiting for it. The basic technical difficulty lies in the fact that the tokens are undistinguishable and seeing a token might

[7] An alternative approach of leaving W_b has synchronization problems: Updating W_b means making it smaller, which would be unsafe if the other agent is reading the message in the same time. This can be solved by leaving $n/2 - W_b$ as a message, as W_b is always at most $n/2$.

mean very different things, depending on the context. Careful case analysis is applied, moreover the algorithm crucially relies on the FIFO requirement and the atomicity of actions (so the agents cannot overtake each other on a link or in a node).

3.2 Detailed Description

The following variables(local to the agent) are maintained by each agent throughout the execution:

- *Steps* – the size of the remaining work assignment
- *HBpos* and *LSPpos* – relative position with respect to the HB and the LSP of the agent; this allows the agent to know when it is at the HB or its LSP without using tokens for marking them
- *MsgILeft* – the last message left by this agent
- *DistC* – used to calculate the distance between LSPs
- *Msg4Me* – the message left for me by the other agent

The tokens are used to mark dangerous ports during cautious walk (a token on a port), as *end-of-message* marks (a token in the middle of a node), and in the HB to limit the number of active agents to two. As the number of possible situations at the HB is quite high, the following careful encoding at the HB is used to limit the number of tokens used: (the triplet represents the number of tokens on the left port, middle, and right port of the HB, respectively):

- $(0, 0, 2)$ – the right port is dangerous (i.e. the HB is the LSP of the right agent), the left port is unexplored (the second agent has not woken-up yet)
- $(0, 0, 1)$ – the right port is safe (the LSP of the right agent already moved to the right), the left port is unexplored
- $(2, 1, 0)$ – both the right and the left port are dangerous, there is a message 0 waiting for the right agent (the second agent woke up before the first agent explored its first port, in the initial step it would have changed $(0, 0, 2)$ to $(2, 0, 1)$, but then immediately in SEEKING transformed that to $(2, 1, 0)$),
- $(1, 0, 1)$ – the right port is dangerous, the left port is safe
- $(2, 0, 0)$ – the left port is dangerous, the right port is safe
- $(1, 0, 0)$ – both ports are safe
- $(1, 1, 0)$ – if seen by the right agent whose LSP is the HB: the left port is safe, there is message 0 waiting for me; if seen by the left agent whose LSP is the HB: the right port is safe, there is message 0 waiting for me
- other configurations do not occur

Note that although configuration $(1, 1, 0)$ has two possible meanings, there will never be confusion as it does not occur in the case where the HB is LSP for both agents (at least one agent must have finished its assignment in order to leave a message).

The algorithm is described for the right agent a. The algorithm for the left agent b is almost identical, the only differences are using opposite directions, and using the floor function instead of ceil when calculating the work assignment.

In the initialization step, the elimination technique is used to limit the number of active agents to two, as well as to choose the right agent a and the left agent b. Note that unlike in [5], the initial work assignment of the first agent is the whole unexplored part – we do not want to deal with the case that the first agent explored its part, while the second agent has not started the algorithm yet. When the second agent starts the algorithm, it will seek the first agent to divide the workload based on what remains unexplored at that moment.

The procedure Checking is executed by an agent that finds a notice that a message has been left for it. The agent reads the message by traversing leftward until an end-point marker of the message (a token in the middle of a node) is found. Note that for the first message this means zero leftward moves. Moreover, a message is left only when both agents are active, therefore there is no confusion if the end-of-message marker is found at the HB.

Algorithm 1 INITIALIZATION and CHECKING

```
1: INITIALIZATION:(upon initial wake-up in the HB)
2: if the right port has no token on it then
3:     put two tokens on the right port
4:     execute procedure EXPLORE(n − 1) as the right agent a
5: else if the left port has no token on it then
6:     move one token from the right port to the left port and add additional token
           to the left port
7:     execute procedure SEEKING() as the left agent b
8: else become Passive immediately
9: end if

10: procedure CHECKING
11:     go left until a token is found in the middle of a node u, counting in Msg4Me
            the number of steps
12:     remove the token, return to your LSP and put the token on the right port
13:     for (i = 0; i ≤ Msg4Me; i++) do   // compute the new work assignment
14:         Steps = ⌈Steps/2⌉   // again floor by the left agent
15:     end for
16:     execute EXPLORE(Steps)
17: end procedure
```

In procedure EXPLORE an agent explores its work assignment using Cautious Walk, checking in the progress for messages from the other agent.

In procedure SEEKING the agent determines the distance between the LSPs and either locates the BH (if there is single unexplored node remaining) or leaves/updates the message for the other agent.

[8] The number of tokens there might have changed from 2 to 1 if u is the HB and the second agent had waken-up meanwhile, but that still does not mean a message notification. Note that this is a place where the code for the left agent is not simple translation of the code for the right agent. The test by the left agent will be: there

Algorithm 2 EXPLORE, SEEKING and CHECK&SPLIT

```
 1: procedure EXPLORE(Steps)
 2:     while true do
 3:         // might enter the BH in this step
 4:         go from the current node u to its right neighbour v
 5:         return back to node u   // doing the Cautious Walk here
 6:         if there is a token on the right port of u[8] then   // no message for me yet
 7:             remove one token from the right port of u
 8:             move to v, put a token on the right port of v and decrement Steps.
 9:             if Steps = 0 then
10:                 // finished exploring my assignment, now find the other agent
11:                 exit the loop and execute procedure SEEKING()
12:             end if
13:         else // there is no token on the right port, i.e. message waiting for me
14:             exit the loop and execute procedure CHECKING()
15:         end if
16:     end while
17: end procedure

18: procedure SEEKING
19:     go left until a token is found at node u, counting in DistC the distance travelled
20:     if found a token on the left port of a non-HB node, or two tokens on the left
            port of the HB then
21:         // u is the LSP of the other agent, leave a message 0
22:         move a token from the left port to the middle of u
23:         execute CHECK&SPLIT(DistC)
24:     else if found one token in the middle of a node u then
25:         // this is the endpoint of my last message, agent b did not read it yet
26:         // update/increment the message
27:         move the token one step to the right and increment MsgILeft
28:         execute CHECK&SPLIT(DistC + MsgILeft −1)
29:     else if found one token on the left port of the HB then   // HB, ignore
30:         ignore and continue on l.19 as if nothing found
31:     end if
32: end procedure

33: procedure CHECK&SPLIT(Dist)
34:     if Dist = n − 2 then   // single unexplored node remaining
35:         the BH is in the remaining unexplored node, terminate
36:     else
37:         return to your LSP and execute Explore(⌈(n − Dist)/2⌉)
38:     end if
39: end procedure
```

3.3 Correctness and Complexity Analysis

Lemma 3. *At most two agents will become active and put a token somewhere.*

Proof. By construction: The second agent to wake-up places two tokens on the left port of the HB (line 6 of INITIALIZATION) and one of these tokens stays there for the whole execution. Due to line 8 no other agent becomes active.

Lemma 4. *If there is a message left for an agent, the agent detects the presence of the message and correctly computes its contents.*

Proof. We prove the lemma for the agent a exploring to the right. The proof for agent b is analogous. When a finds out that there is a message waiting for it (by not finding its token when returning from cautious walk, lines 13-14 of EXPLORE), according to line 11. of CHECKING it travels to the left to locate the end-of-message token placed in the middle of a node. Due to FIFO and atomicity, it cannot overtake the agent b laying the message, i.e. it will always find the end-of-message token (even if b is concurrently incrementing the message). Moreover, the searching for the end-of-message token is safe, i.e. a will not travel past b's LSP: A non - 0 message is left only by an agent that already explored its assignment, i.e. the distance between LSPs must be at least $\lfloor n/2 \rfloor$.

Let us define the *work assignment* of the right/left agent as follows:

- If the agent is traversing the dangerous link from its LSP and there is a message waiting for it, the work assignment of the agent is only the node on the other side of the dangerous port
- Otherwise, the work assignment of the agent is the *Steps* nodes to the right/left of agent's LSP.

Lemma 5. *At any moment, the work assignments of the right and left agents are disjoint. Moreover, if there is no message waiting for an agent then the work assignments form a partition of the part of the ring delimited by the LSPs (or the LSP of the right agent and the* HB*, if there is single agent active) and not containing the* HB.

Proof. By induction over the execution. According to line 4 in INITIALIZATION, the initial work assignment of the right agent a covers all nodes between the HB and a's LSP (which is the HB). This property is maintained by construction of EXPLORE, until the second agent b leaves a message for a.

If there is a message waiting for an agent (say a), the agent b at the moment it left the message computed its work assignment as half of the part remaining unexplored between the LSPs. Since this part contains at least two nodes (otherwise b would terminate), half of it (= b's assignment) does not contain the node a is currently heading to.

---

is a token on the left port of u, or u is the HB and there are two tokens on the left port of u.

Finally, if there are no messages waiting, it is either at the very beginning when there is only one agent (already dealt with) or after one agent (say a) read a message m and recomputed its *Steps* according to lines 13-14 of CHECKING. Consider the value d of variable *DistC* of agent b at the moment when it left the first message (containing 0) for agent a. Since b has just finished its assignment, by induction hypothesis the current value of a's *Steps* equals to $n - d$ (i.e. the value b started with halving). At the moment a reads the message m, b had halved its Steps $m+1$ times (line 37 in CHECK&SPLIT), and that is exactly what a does in lines 13-14 of CHECKING (also applying Lemma 4). The partitioning works properly even when halving odd-sized workloads because one agent uses ceiling and the other uses floor. Afterwards, the invariant of the lemma is maintained by construction of *Explore* until another message is left.

Since, by construction, the only previously unexplored nodes an agent enters are in its work assignment, we immediately get:

Corollary 1. *At most one agent enters the* BH.

Another consequence of Lemma 5 is that at any moment there is at most one message present, and at most one agent executing SEEKING or CHECK&SPLIT.

From construction (line 37 of CHECK&SPLIT and lines 13-16 or CHECKING) it follows that the parameter STEPS is at least halved in each consecutive call to EXPLORE (i.e. the unexplored area is at least halved). Since no waiting is specified in any place of the algorithm; SEEKING, CHECK&SPLIT and CHECKING each take at most $n - 2$ moves and either terminate or are followed by a call to EXPLORE, the algorithm terminates in $O(n \log n)$ steps. As the only way to terminate is to detect there is a single unexplored node, this node must contain the BH.

Note that at any moment at most three tokens are present in the system: one remaining at the HB and one by each agent used for marking the dangerous port and messaging. This follows from:
- the first agent puts two tokens (line 3 of INITIALIZATION), the second agent adds one more token (line 6) and no more agents become active (line 8)
- at any other place of the algorithm an agent puts a token that it has previously removed from some other place (moving the dangerous port mark, leaving a message, incrementing a message or reading a message).

Putting all together we obtain:

Theorem 1. *Algorithm Divide with Token correctly locates the* BH *employing two agents and three tokens in total, spending* $O(n \log n)$ *moves.*

4 Conclusions

In this paper we answered the following question: *Is it possible to locate a black hole in an asynchronous ring network using tokens instead of whiteboards?*

We answered this question affirmatively, showing that this can actually be achieved without any loss of performance. In fact, we present two algorithms,

both using $O(1)$ tokens, that allow a team of *two* agents to determine the location of the black hole with $\Theta(n \log n)$ moves.

This result constitutes is a significant improvement over previous results, which used both stronger communication media (whiteboards) and more memory in the network ($O(n)$, resp. $O(\log n)$ after some modifications).

The second protocol presented here uses only three tokens in total, while still maintaining the optimal cost of $\Theta(n \log n)$ moves. An intriguing question is whether it is possible to further reduce the number of tokens to 2, and what would be the cost in such case.

References

1. L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Election and rendezvous in fully anonymous systems with sense of direction. *Theory of Computing Systems*, 2006. To appear. Preliminary version in Proc. of SIROCCO 2003.
2. D. M. Chess. Security issues in mobile code systems. In *Proc. Conf. on Mobile Agent Security*, LNCS 1419, pages 1–14, 1998.
3. J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in tree networks. In *Proc. 8th International Conference on Principles of Distributed Systems (OPODIS 2004)*, pages 35–45, 2004.
4. S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, and N. Santoro. Optimal search for a black hole in common interconnection networks. *Networks*, 2006. To appear. Preliminary version in Proc. of OPODIS 2002.
5. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 2006. To appear. Preliminary version in Proc. of DISC 2001.
6. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in a arbitrary networks: optimal mobile agent protocols. *Distributed Computing*, 2006. To appear. Preliminary version in Proc. of PODC 2002.
7. S. Dobrev, P. Flocchini, and N. Santoro. Improved bounds for optimal black hole search in a network with a map. In *Proc. of 10th International Colloquium on Structural Information and Communication Complexity*, pages 111–122, 2004.
8. P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. In *6th Latin American Theoretical Informatics Symp.*, pages 141–151, 2004.
9. P. Fraigniaud and D. Ilcinkas. Digraph exploration with little memory. In *21st Symp. on Theoretical Aspects of Computer Science*, pages 246–257, 2004.
10. M.S. Greenberg, J.C. Byington, and D. G. Harper. Mobile agents and security. *IEEE Commun. Mag.*, 36(7):76 – 85, 1998.
11. F. Hohl. A model of attacks of malicious hosts against mobile agents. In *Proc. of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems*, LNCS 1603, pages 105 – 120, 1998.
12. R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary graphs. In *Proc. 12th Coll. on Structural Information and Communication complexity (SIROCCO'05)*, pages 200–215, 2005.
13. R. Oppliger. Security issues related to mobile code and agent-based systems. *Computer Communications*, 22(12):1165 – 1170, 1999.
14. T. Sander and C. F. Tschudin. Protecting mobile agents against malicious hosts. In *Proc. of Conf on Mobile Agent Security*, LNCS 1419, pages 44–60, 1998.