International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 4, Number 1, pages 151–173, January 2014

Decontaminating a Network from a Black Virus

Jie Cai
School of Computer Science
Carleton University
Email: jie_cai@carleton.ca

Paola Flocchini
School of Electr. Eng. and Comp. Science
University of Ottawa
Email: flocchin@site.uottawa.ca

and

Nicola Santoro
School of Computer Science
Carleton University
Email:santoro@scs.carleton.ca

Received: July 26, 2013
Revised: October 29, 2013
Accepted: November 29, 2013
Communicated by Susumu Matsumae

Abstract

In this paper, we consider the problem of decontaminating a network from a *black virus* (BV) using a team of mobile system agents. The BV is a harmful process which, like the extensively studied black hole (BH), destroys any agent arriving at the network site where it resides; when that occurs, unlike a black hole which is static by definition, a BV moves, spreading to all the neighbouring sites, thus increasing its presence in the network. If however one of these sites contains a system agent, that clone of the BV is destroyed (i.e., removed permanently from the system). The initial location of the BV is unknown a priori. The objective is to permanently remove any presence of the BV from the network with minimum number of site infections (and thus casualties). The main cost measure is the total number of agents needed to solve the problem.

This problem integrates in its definition both the harmful aspects of the classical *black hole search* problem (where however the dangerous elements are static) with the mobility aspects of the classical *intruder capture* or *network decontamination* problem (where however there is no danger for the agents). Thus, it is the first attempt to model mobile intruders harmful not only for the sites but also for the agents.

We start the study of this problem by focusing on some important classes of interconnection networks: *grids*, *tori*, and *hypercubes*. For each class we present solution protocols and strategies for the team of agents, analyze their worst case complexity, and prove their optimality.

Keywords: Network decontamination, Dangerous graphs, Mobile Entities, Distributed Computing

1 Introduction

1.1 Framework and Problem

Mobile agents are widely used in distributed and networked systems; however, the support of mobile agents can also cause security issues and threats to the network. In particular, a malicious agent can cause computer nodes to malfunction or crash by contaminating or infecting them; additionally, a contaminated or infected host in turn can destroy working agents for various malicious purposes. In the distributed computing literature, the former situation is categorized as one of *harmful agents*, the latter as one of *harmful hosts* (e.g., see [30]).

In the case of a harmful host, the primary concern has been on locating its position, to isolate and later deactivate it; the theoretical focus has been on a particularly harmful host, called *black hole*: a network node infected by a process which destroys any incoming agent without leaving any detectable trace of the destruction. Indeed, the problem of locating a black hole, called *black hole search* (BHS), has been extensively studied (e.g., [7, 8, 9, 11, 12, 17, 19, 20, 33, 37, 38, 45]). It is worth to point out that a black hole is a presence which is harmful to agents but it is *static*, that is, it does not propagate in the network and so it is not harmful to other sites.

The theoretical work related to harmful agents has focused on the problem called *intruder capture* (IC) (also known as *graph decontamination* and *connected graph search*): an extraneous mobile agent, the intruder, moves through the network infecting the visited sites; the task is to decontaminate the network using a team of system agents avoiding recontamination. This problem has been extensively studied (e.g.,[3, 4, 5, 14, 16, 22, 23, 28, 29, 32, 35, 36, 40, 41, 42, 43, 44, 46]). Let us point out that, in this problem, the harmful presence (the intruder) is mobile and harmful to the network sites, but does not cause any harm to the system agents.

Summarizing, the previous studies on BHS have not considered the transition characteristics of the harmful nodes: black holes do not move and their number does not change; similarly, the previous studies on IC have not considered the case of a mobile intruder harmful for both sites and agents. The problem we consider in this paper combines some aspects of black hole search with some of network decontamination: the harmful process is mobile (like an intruder) and harmful also to the system agents (like a black hole).

The harmful presence we consider is a *black virus* (BV), a dangerous process initially resident at an unknown network site. Like a black hole, a BV destroys any arriving agent. When this occurs, unlike a black hole, the BV moves spreading to all neighbouring sites, thus potentially increasing its number, presence and damage in the network. On the other hand, the original node containing the BV becomes empty and clean. A BV is destroyed when it moves to a node that contains an anti-viral system agent; in this case, the agent is able to deactivate and permanently remove that instance of the BV. The problem we study is that of *black virus decontamination* (BVD), that is to permanently remove any presence of the BV from the network using a team of system agents. Solving this problem is dangerous for the agents, since any agent arriving at a node where an instance of the BV resides will be destroyed; it is obviously dangerous for all the nodes where the BV will spread to. A protocol defining the actions of the agents solves the BV decontamination problem if, within finite time, at least one agent survives and the network is free of BVs. A solution protocol will specify the strategy to be used by the agents; that is, it specifies the sequence of moves across the network that will enable the agents, upon all being injected in the system at a chosen network site, to decontaminate the whole network. The goal of a solution protocol is to minimize the *spread* of the BV i.e., the number of node infections by the BVs; note that, since each instance of the BV has to be eventually removed and each removal requires the destruction of at least one agent, the spread also measures the number of agent *casualties*. The other important cost measure is the *size* of the team, i.e. the number of agents employed by the solution. An additional cost measure is the number of moves required.

A desirable property of a decontamination protocol is to protect the nodes which have been already explored by mobile agents from being infected or contaminated by the BV spreading; note that re-contamination of a decontaminated site will re-occur if the virus is able to return to that site in the absence of an agent. Following the literature on classical network decontamination (e.g.,

see [31]), a solution protocol with such a property will be called *monotone*.

1.2 Main Contributions

In this paper we define the *black virus decontamination* (BVD) problem and start the study of its solution.

We first establish some basic facts. In particular, we observe that *monotonicity* is necessary for optimality: in every network, to minimize the spread, a solution protocol must be monotone. We also describe a general solution strategy for the problem.

We then study in detail the black virus decontamination problem for three large classes of common network topologies: (multi-dimensional) *grids*, (multi-dimensional) *tori* and *hypercubes*.

For each class, we design a monotone solution protocol which decontaminates the networks regardless of the number of dimensions, and we analyze its complexity. All the protocols are *optimal* both in terms of spread (i.e., number of casualties) and of total number of agents, and are asymptotically optimal in the total number of moves; a summary of the results are shown in the table below, where n and m denote the number of nodes and of links, respectively.

Network	Spread	Size	Moves
q-Grid	$q+1$	$3q+1$	$O(m)$
q-Torus	$2q$	$4q$	$O(m)$
Hypercube	$\log n$	$2\log n$	$O(m)$

Table 1: Summary of results.

In all our solutions, the agents perform local computations only based on their current location; an agent's local memory containing just $O(\log n)$ bits is then sufficient to perform the BV decontamination.

1.3 Related Work

The black virus decontamination problem we study is related to two problems extensively studied in the literature: black hole search and intruder capture.

In *black hole search* (BHS), there is a harmful process, the *black hole*, stationary at one or more network nodes; a black hole destroys any agent arriving at the node where it resides without any detectable trace. The task is for a team of system agents to locate the black hole(s); this task is dangerous for the agents, and it is completed when at least one agent survives and reports the location(s) of the black hole(s). The black hole search problem has been originally studied in ring networks [20] and has been extensively investigated in various settings since then. The main distinctions made in the literature are whether the system is *synchronous* or *asynchronous*. The majority of the work focuses on the asynchronous model. The problem has been studied in *rings* [20], common interconnection networks [17], directed graphs [11], and networks of arbitrary topologies under a variety of assumptions on the agents' knowledge [19, 33]. In asynchronous settings, a variation of the model where communication among agents is achieved by placing tokens on the nodes, has been investigated in [18, 21, 24, 45]. The case of black links in arbitrary networks has been studied in [6, 26], respectively for anonymous and non-anonymous nodes. The study of BHS in *time-varying graphs* has been started in [25]. In synchronous networks, where movements are synchronized and it is assumed that it takes one unit of time to traverse a link, the techniques and the results are quite different. Tight bounds on the number of moves have been established for some classes of *trees* [13]. In the case of general networks finding the optimal strategy is shown to be NP-hard [12, 37], and approximation algorithms are given in [37, 38]. The case of multiple black holes has been investigated in [9], and tight bounds are given. The impact of synchronicity in *directed graphs* has been studied in [39]. Recent investigations have also dealt with scattered agents searching for a black hole in *rings* and *tori* [7, 8]. Finally, the problem of repairing networks with black holes has been recently investigated [10, 15].

In the *intruder capture* problem, an intruder moves from node to node at arbitrary speed through the network, contaminating the sites it visits; a team of agents has to capture the intruder; the intruder is captured when it comes in contact with an agent. This problem, first introduced in [4], is equivalent to the problem decontaminating a network infected by a virus while avoiding any recontamination. This problem, also called *connected graph searching*, is a non-trivial variant of the well known *graph searching* problem, which has been extensively studied (see [31] for a recent comprehensive survey), and is closely related to some basic graph parameters and concepts, including tree-width, cut-width, path-width, graph minor, etc. The intruder capture problem has been investigated for a variety of network classes: *trees* [3, 4, 16], *hypercubes* [22], *multi-dimensional grids* [28], *pyramids* [44], *chordal rings* [23], *tori* [23], *outerplanar graphs* [32], *chordal graphs* [43], *Sierpinski graphs* [42], *star graphs* [36], *product graphs* [35], graphs with large clique number [46], while the study of arbitrary graphs has been started in [5, 34]. The study of the problem with stronger requirements for recontamination has been done for *toroidal meshes* and *trees* with strong majority threshold [41], and for *grids*, *tori* and *hypercubes* with arbitrary threshold [40, 27].

2 Model and Terminology

The environment in which the agents operate is a network whose topology is modelled as a simple undirected connected graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ links. For a given graph $G = (V, E)$ we denote by $E(v) \subseteq E$ the set of edges incident on $v \in$ V, by $d(v) = |E(v)|$ its degree, and by $\triangle(G)$ (or simply $\triangle$) the maximum degree of G.

The network is not anonymous: every node has a distinct *id*, visible to the agents visiting it. The links incident to a node are labelled with distinct port numbers. The labelling mechanism could be totally arbitrary among different nodes. Without loss of generality, we assume the link labelling for node v is represented by set $l_v : (v, u) \in E : u \in V$ and $l_v = 1, 2, 3, ..., d(v)$.

In the network operate a system of mobile agents. An agent is modelled as a computational entity which can move from node to neighbouring node. Agents may have different functions/roles (i.e. states). In the following, for simplicity of presentation, we assume that it takes one unit of time for an agent to move from one node to another, and that computing and communication time is assumed to be negligible compared to moving time. In other words, we assume that the network is synchronous. However all the results established here hold also if the system is asynchronous, i.e. the time of each activity (processing, communication, moving) is finite but otherwise unpredictable; in particular, all the solution algorithms can be easily adapted to the asynchronous case.

More than one agent can be at the same node at the same time. Communication among agents (e.g. to coordinate, synchronize or update their maps, and etc.) occurs when they are at the same node. Each agent A_i has a unique id from some totally ordered set. The agents all execute the same protocol.

In G there is a node infected by a *black virus* (BV), an extraneous harmful process endowed with reactive capabilities for destruction and spreading. The location of the BV is not known a priori. It is harmful not only to the node where it resides but also to any agent arriving at that node. In fact like a black hole, a BV destroys any agents arriving at the network site where it resides. When that occurs, (unlike a black hole which is static by definition) the BV clones itself and moves from the current node spreading to all the neighbouring sites leaving the current node clean; upon arriving at a node, (the clone of) the BV infects the node and stays inactive until an agent arrives. As the clones of a BV have the same capabilities of the original BV, the number of BVs in the network may increase over time. However, at each node at any time there is at most one BV; multiple copies of the BV (arriving) at the same node merge. A BV is destroyed if there is already an agent at a node when the BV arrives. Thus, the only way to eliminate a BV from the system is to surround it completely and let an agent attack the BV by moving to the BV node. In this case, the node where the BV resides is cleaned and all the generated clones of that BV are destroyed.
Summarizing, there are four possible situations:
Situation 1. Agents arrive at a node v which is empty or where there are other agents: the agents can communicate with each other.

Situation 2. Agents arrive at a BV node v: they are destroyed; the BV moves to each of v's neighbours; v becomes clean.
Situation 3. BVs arrive at v which is empty or where there is a BV: the BVs merge and v is a BV node.
Situation 4. BVs arrive at a node v where there are agents: the BVs are destroyed.

The *BV decontamination problem* is to permanently remove any presence of the BV from the network using a team of agents. The initial location of the BV is unknown. A protocol defining the actions of the agents solves the BV decontamination problem if, within finite time, at least one agent survives and the network is free of BVs. A solution protocol is *monotone* if no recontamination occurs in any of its executions; that is, once a node is explored (or decontaminated), it will never be (re)contaminated. Notice that determining the initial location of the BV can be achieved only by having an agent arriving there; the cost of this operation will be the destruction of the agent, and the creation of more BVs which will move to the neighbouring nodes; unless these neighbouring nodes are protected by a resident agent, they will become BV nodes.

The goal of a solution protocol is to minimize the *spread* of the BV i.e., the number of node infections by the BVs; note that, since each instance of the BV has to be eventually removed and each removal requires the destruction of at least one agent, the spread also measures the number of agent *casualties*. We denote by $spread_P(G)$ the number of casualties incurred in the worst case (i.e., over all possible initial locations of the BV) when executing solution protocol P in G, and by $spread(G)$ the minimum over all solution protocols. We denote by $size(G)$ the smallest team size capable of decontaminating G with optimal spread in the worst case (i.e., over all possible initial locations of the BV). An additional cost measure is the number of moves performed by the agents; when measuring it for our protocols we will consider synchronous execution.

3 Basic Properties and Algorithmic Tools

3.1 Basic Properties and Bounds

Our concern is with solution protocols that are efficient and possibly optimal with respect to number of casualties, the *spread*, and the total number of agents, the *size*. In this respect, the first observation is with regards to *monotonicity* of the solution.

Property 3.1 *Monotonicity is necessary for spread optimality*

Proof: By contradiction, let P be a protocol that solves BVD in G non-monotonically with optimal spread. That is, during the execution of P in G, a BV moves to an unguarded node v which was previously explored by an agent. As a result, v becomes a *BV* node. Note that the move of the BV to v must have been triggered by an agent A moving to a BV node u neighbour of v. Consider now a protocol P', whose execution is totally identical to that of P except that, before A moves to u, an agent B moves to v; B is either an agent that, at the time of A moving to v, is not being used by P for protection of an explored neighbour of v, or an additional one. This means that when the move of A to u causes the BV to move to v, agent B destroys that BV without harm for itself. In other words $spread_{P'}(G) < spread_P(G)$ contradicting the spread optimality of P. □
Hence, we can restrict ourselves to designing monotone solution.

The values of the *size* and of the *spread* depend on many factors, first and foremost the structure of the graph, but also on the location of the home-base and of the BV. Some basic bounds follow immediately from the definition of the problem. In particular, regardless of the topology of the network, since at least one agent must survive by definition, we have

Property 3.2 *For any network G, $size(G) \geq spread(G) + 1$.*

Let v be an articulation point of G, let $G_{min}(v)$ denote the *smallest* (in terms of number of nodes) connected component formed by removing v and its incident edges from G, and let $\rho_{min}(v)$ denote its size.

Property 3.3 *If G has an articulation point v, then $spread(G) \geq n - \rho_{min}(v)$.*

Proof: Consider the case when the home-base is a node u in $G_{min}(v)$, and the BV is located at the articulation point v. In this case, regardless of the solution strategy, the BV will spread to all nodes of all connected components formed by removing v from G, with the exception of $G_{min}(v)$. □

Let $e = (u, v)$ be a cut edge of G, let $G_{min}(e)$ denote the *smallest* (in terms of number of nodes) of the two connected components formed by removing e from G, and let $\rho_{min}(e)$ denote its size; without loss of generality let u be in $G_{min}(e)$.

Property 3.4 *If G has a cut edge e, then $spread(G) \geq n - \rho_{min}(e)$.*

Proof: Let $e = (u, v)$ be a cut edge; without loss of generality let u be in $G_{min}(e)$. Consider the case when the home-base is a node in $G_{min}(e)$, and the BV is located at v. In this case, regardless of the solution strategy, the BV will spread to all nodes of $G \setminus G_{min}(e)$. □

As a corollary of Property 3.4, it follows that there are graphs where the *size* and the *spread* are $\Omega(n)$:

Corollary 3.5 *Let G contain a node v of degree $|E(v)| = 1$. Then, $spread(G) \geq n - 1$.*

Notice the the class of graphs covered by Corollary 3.5 includes acyclic networks, i.e. *trees*. In the case of trees, the bounds are tight:

Lemma 3.1 *Let T be a tree on n nodes. Then, $size(T) = spread(T) + 1 = n$.*

Proof: Necessity is by Property 3.2 and Corollary 3.5. To see that n agents always suffice in a tree T, consider a depth-first traversal where, starting from the homebase, a single agent moves forward; once the outcome is determined (either the agent returns or a BV arrives), all remaining agents move to that node; when backtracking, all remaining agents move back. It is easy to see that this strategy decontaminates the tree with at most $n - 1$ casualties. □

In other words, there are graphs (e.g., trees) where the *size* and the *spread* are $\Theta(n)$. On the other hand, there are networks where the *spread* and the *size* are $\Theta(1)$ regardless of the number of nodes. To see this, let us concentrate on *regular* graphs.

Property 3.6 *Let G be a Δ-regular graph. Then $spread(G) \geq \Delta$.*

Proof: Since the very first node v explored by the agents can be a BV, the number of casualties is at least one for each neighbour of v (other than the homebase) plus the one incurred at v. □

In most regular graphs, the relationship between *size* and *spread* can be much higher than that expressed by Property 3.2; in fact

Property 3.7 *Let G be a triangle-free Δ-regular graph. Then $size(G) \geq 2\Delta$.*

Proof: Let the first node v explored by the agents be a BV; thus all $\Delta - 1$ neighbours (other that the homebase) of v become BV nodes. Since G is triangle-free, none of these BV nodes are neighbours. This means that each of these BV nodes has now Δ clean neighbours (including v). Let z be the first of these BV nodes to be cleared; at least Δ agents are needed to protect the clean neighbours of z, in addition to the Δ casualties, for a total of at least 2Δ agents. □

Notice that the class of graphs covered by Property 3.7 includes among others *rings*, *tori*, and *hypercubes*. In the case of *rings*, both *size* and *spread* are constant, independent of n, and the bounds are tight:

Lemma 3.2 *Let R be a ring of n nodes. Then, $size(R) = 4$ and $spread(R) = 2$*

Proof: By Property 3.6 it follows that $spread(R) \geq 2$ and thus, by Property 3.7, $size(R) \geq 4$. Sufficiency of four agents with two casualties regardless of the ring size n is immediate. □

3.2 Algorithmic Structure and Tools

In this paper we concentrate on the specific case of multi-dimensional *grids*, multi-dimensional *tori*, and *hypercubes*; we develop optimal protocols for all these classes.

All our protocols consist of two separate phases. In the first phase, the network is traversed until the initial location of the BV is determined. When this occurs, that location is cleared but all its unprotected neighbours have become BV nodes. In the second phase, these BV nodes are permanently removed ensuring that none of their safe neighbours are infected.

In the first phase, the solutions use two basic techniques, safe-exploration and shadowing, explained below.

Safe-exploration. To minimize the spread of the virus, during the traversal, the actual exploration of a target node v from a safe node u is performed using a *safe-exploration* technique. This technique is executed by two agents, the *explorer agent* EA, and the *leader explorer agent* LEA both residing at u. The exploring agent EA goes to v to explore it. If EA returns both agents proceed to v. If instead v was a BV node, LEA becomes aware of that because a BV arrives through that link instead of EA; in this case, the second phase starts.

Shadowing. To insure monotonicity, at any point in time the already explored nodes must be protected so that they do not come into contact with a moving BV. To achieve that, during the traversal by EA and LEA, some agents are employed to guard the already visited neighbours of the node currently under exploration. This technique is called *shadowing*. In the following, we use the term *shadowing agent* (SA) to indicate an agent who plays the shadowing role to prevent re-contamination of the explored nodes. Note that LEA, in safe-exploration, also plays the role of a shadowing agent.

Once the BV node is detected, the protocols enter the second phase. In this phase, the SAs sequentially "surround" the newly created BVs (i.e., an agent is deployed in each one of its safe neighbours). Once a BV is surrounded, an agent moves to it to remove it; this agent dies, permanently destroying that BV. This process is repeated until all instances of the BV are permanently removed.

In some of the protocols (specifically, for meshes and tori), the number of agents surviving the first phase is smaller than what is needed in the second phase. It is intended that these extra agents travel with LEA during the first phase; thus, they are co-located with LEA when the second phase starts.

The correct behaviour of the algorithms relies on the proper synchronization of actions of all the agents; in particular, to ensure monotonicity, the exploration of a new node u by the Explorer can only start after the already explored neighbours of u are covered by Shadowing Agents. All the proposed protocols, irrespective of the topology of the graph, use the same coordination mechanisms, the only difference being on whether the system is synchronous of asynchronous. These mechanisms are rather straightforward and are described in the Appendix.

4 BV Decontamination of Grids

4.1 Base Case: 2-Dimensional Grid

The general algorithm for BV decontamination of multi-dimensional grids makes use of the basic algorithm for the 2-dimensional case presented and analyzed in this section. Let G be a 2-dimensional grid of size $d_1 \times d_2$; without loss of generality, let the nodes of G be denoted by their column and row coordinates (x, y), $1 \leq x \leq d_1, 1 \leq y \leq d_2$.

Agents LEA and EA start from the corner node $(1, 1)$ to perform a safe exploration of G with shadowing to locate the BV while protecting the nodes already visited. Once triggered, the BV spreads to its unprotected neighbours, and the elimination phase starts where agents are deployed to surround each newly created BV and to eliminate them.

The safe exploration proceeds along the columns of the grid in a snake-like fashion (see the dotted route in Figure 1), where odd columns are traversed downwards and even columns are traversed upwards. EA leads the exploration, while LEA stays one node behind to detect the presence of

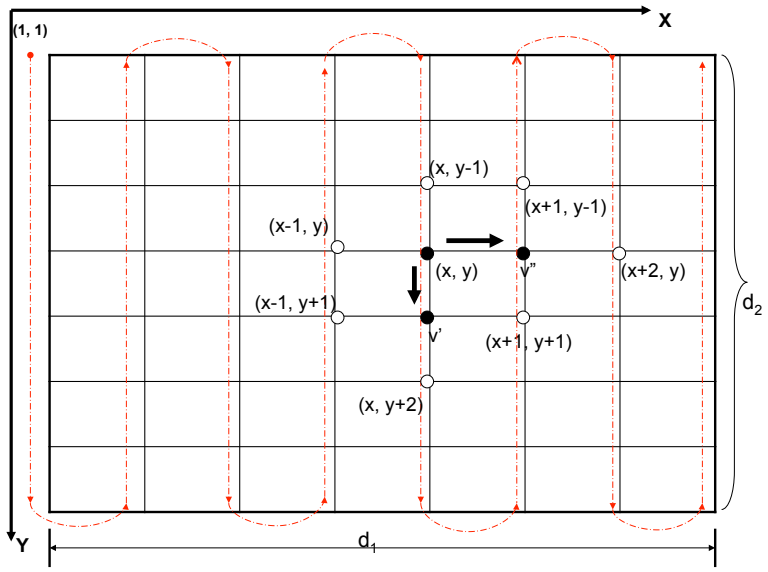

Figure 1: BV exploration in 2D

the Black Virus, should it be triggered by EA. While the safe exploration proceeds, an appropriate number of shadowing agents are deployed to cover the already explored nodes that are neighbours of the node under exploration by EA so that when the BV is triggered, those nodes are protected and will not be recontaminated.

Let $v = (x, y)$ be the node under exploration, with $1 \leq x \leq d_1$ and $1 \leq y \leq d_2$. Let x^+ and x^- be defined as follows: $x^+ = x + 1$ (if $x \neq d_1$), $x^+ = x$ otherwise. Similarly: $x^- = x - 1$ (if $x \neq 1$), $x^- = x$ otherwise. An analogous definition holds for y^+ and y^-.

We can now define the set of explored and unexplored neighbours of node $v = (x, y)$:

$$N_{ex}(v) = \begin{cases} \{(x^-, y), (x, y^-)\} & \text{if } x \text{ is odd} \\ \{(x^-, y), (x, y^+)\} & \text{if } x \text{ is even} \end{cases}$$

$$N_{un}(v) = \begin{cases} \{(x^+, y), (x, y^+)\} & \text{if } x \text{ is odd} \\ \{(x^+, y), (x, y^-)\} & \text{if } x \text{ is even} \end{cases}$$

In a step of the shadowed exploration, the actions performed are as follows:

ALGORITHM *BVD-2G*: SHADOWED EXPLORATION

Agents EA and LEA are at safe node (x, y).
1. Agents compute $Next(x, y)$:
   If x is odd and $x \neq d_1$, $y \neq d_2$: $Next(x, y) = (x, y^+)$
   If x is even and $x \neq d_1$, $y \neq 1$: $Next(x, y) = (x, y^-)$
   If x is odd and $y = d_2$: $Next(x, y) = (x^+, y)$
   If x is even and $y = 1$: $Next(x, y) = (x^+, y)$
2. SAs move to occupy the nodes $N_{ex}(Next(x, y)) \setminus (x, y)$
3. EA moves to $Next(x, y)$.
4. If unarmed, EA returns to (x, y) and both EA and LEA move to $Next(x, y)$.

The shadowed exploration continues until EA meets the BV, at which time EA dies, LEA detects the presence of the BV (i.e., it receives a new virus from the node v just explored by EA), and the

elimination phase starts.

At this point LEA, together with the shadowing agents fully cover $N_{ex}(v)$, and new BVs have spread to $N_{un}(v)$.

ALGORITHM *BVD-2G*: SURROUNDING AND ELIMINATION

BV originally in v now spread to $N_{un}(v)$
LEA and SAs are at the nodes $N_{ex}(v)$

1. LEA moves to v.
2. For each $u \in N_{un}(v)$:
   2.1 Agents move to each unoccupied $z \in \{N(u)\}$
   2.2 An agent moves to u to permanently remove that BV.

Note that while the two exploring agents traverse the first column (i.e., $x = 1$), no other shadowing is necessary; when they traverse subsequent columns (i.e., $x > 1$), a shadowing agent moves to the already explored neighbour (x^-, y) so to protect it, should the node (x, y) to be explored next contain a BV.

Theorem 4.1 *Protocol* BVD-2G*, performs a BV decontamination of a 2-dimensional Grid using $k = 7$ agents and 3 casualties.*

Proof: Let $v = (x, y)$ be the node containing the BV. When EA moves to v, it will be destroyed and the BV will move to all neighbours of v. When this happens, according to the algorithm, LEA is protecting the neighbour from which EA moved to v. If $x > 1$, the neighbour $(x - 1, y)$ is protected by a shadowing agent; if $x = 1$, then we are still in the first column and this neighbour does not exist. So, when the BV moves to the neighbours of v, the explored ones are protected and will not be infected by the BV; this means that the BV can safely move only to the unexplored neighbours of v, of which there are at most two. In other words, after v is explored, at most two BV nodes are formed; furthermore, since the grid is a triangle-free graph, they are not neighbours. This means that the BV nodes can be sequentially and separately surrounded and destroyed using at most six agents (including SA and LEA): one to enter a BV node and four to protect the neighbours. Hence, in addition to EA, at most two more agents die, and the total number of employed agents is seven. □

It is not difficult to verify that both the spread and the size of Protocol *BVD-2G* are optimal.

Theorem 4.2 *Let G be a 2-dimensional grid. Then, regardless of the number of nodes, $spread(G) = 3$ and $size(G) = 7$.*

Proof: Since the very first node visited by any algorithm could be a BV, at least two more BV will be generated; that is, for any solution protocol P, $spread_P(G) \geq 3$. Of these two generated BVs at least one has four neighbours; furthermore, since G is triangle-free, these neighbours have no edges in common, and none of them is the other BV node. This means that at least four agents are needed in addition to the three casualties; that is, for any solution protocol P, $size_P(G) \geq 7$. □

Let us now consider the number of moves.

Theorem 4.3 *Protocol* BVD-2G*, performs a BV decontamination of a 2-dimensional grid of size n with at most $9n + O(1)$ moves in time at most $3n$.*

Proof: Let $v = (x, y)$ be the BV node, and let the size of the grid be $n = d_1 \times d_2$.

Let us first consider the number of moves performed during the shadowed exploration. The travelling distance from the home base to v is $d = (x-1)d_1 + x + y$. The shadowing agent follow the same path, except when EA and LEA traverse the first column (that does not need shadowing), so the number of moves for the shadowing agent is bounded by $d - 1$. The number of moves is then $3(d-1)+1$ for EA, $(d-1)$ for LEA, and at most $(d-1)$ for the SA. The other four agents (needed

for the surrounding and elimination) travel with LEA incurring in the same cost for a total of at most $4(d-1)$. We then have an overall cost of at most $9(d-1)+1$ moves for this phase.

Consider now the number of moves performed for Surrounding and Elimination. The new BV nodes v' and v'' are not connected, so they can be surrounded and cleaned sequentially. Each surrounding and cleaning requires a constant number of moves by 5 agents (4 to surround and 1 to clean the BV). Hence $O(1)$ moves are performed in this phase.

In total we then have that the number of moves is at most $9(d-1)+O(1) \leq 9n+O(1)$.

As for the time complexity. The time required for the exploration phase is equal to the number of moves of EA, which is $3(d-1)+1$; the time required for the surrounding and elimination phase is constant. □

4.2 Multi–Dimensional Grid

Let M be a q-dimensional grid of size $d_1 \times \ldots \times d_q$ and let each node of M be denoted by its coordinates $(x_1, \ldots, x_q)$, $1 \leq x_i \leq d_i$. The algorithm, called *BVD-qG*, follows a general strategy similar to the one described in Section 4.1: a safe exploration with shadowing, followed by a surrounding and elimination phase.

We first describe the path followed for the safe exploration, and then the details of the surrounding and elimination phase.

Informally, the multi-dimensional grid is partitioned into $d_1 \times \ldots \times d_{q-2}$ 2-dimensional grids of size $d_{q-1} \times d_q$. Each of these grids is explored using the shadowed traversal technique of Protocol *BVD-2G* described in Section 4.1, in a snake-like fashion column by column, and returning to the starting point; in the following we refer to this exploration as *Traverse2*. From that starting point, the exploration proceeds to another grid, with a neighbouring starting point.

More precisely, given a sequence of $k < q$ integers $i_1, \ldots i_k$ with $1 \leq i_j \leq d_j$ (where $1 \leq j \leq k$), let $M[i_1 \ldots i_k]$ denote the $(q-k)$-dimensional sub-grid of M formed by the elements $\{(i_1, i_2, \ldots, i_k, x_{k+1}, \ldots, x_q)\}$ where $1 \leq x_l \leq d_l$ (where $k+1 \leq l \leq q$). The exploration traversal is achieved using the procedure TRAVERSE(V,K) described below, where V is a sequence of k integers. The symbol $V \circ j$ denotes the operation of adding integer j at the end of sequence V, resulting in V containing now $k+1$ integers. Initially, the procedure is invoked with $V = \emptyset$ and $k = 0$; at each end of the recursion, V is a 2-dimensional grid of size $d_{q-1} \times d_q$.

```
TRAVERSE(V,K)

if (q − k) = 2 then Traverse2(M[V])
else
  for 1 ≤ j ≤ d_{k+1}
      Traverse(V ∘ j, k + 1)
```

Figure 2 visualizes the traversal on a 3-dimensional Grid M with size $d_1 \times d_2 \times d_3$. M is explored by sequentially traversing the 2-dimensional grids $M[1], M[2], \ldots, M[d_1]$.

During the traversal, before exploring a node $v = (x_1, \ldots, x_q)$ $(1 \leq x_i \leq d_i)$ from a node u, the SAs move to the already explored neighbours of v to protect them; note that u is already protected by LEA. The set of these nodes is $N_{ex}(v) = \{(x_1^-, x_2, ..., x_q), (x_1, x_2^-, ..., x_q), ..., (x_1, x_2, ..., x_q^-)\} \setminus \{v\}$, where

$$x_i^- = \begin{cases} x_i - 1 & \text{if } i \leq q-1 \text{ and } x_i > 1 \\ x_i - 1 & \text{if } i = q \text{ and } x_i \text{ is odd and } x_i > 1 \\ x_i + 1 & \text{if } i = q \text{ and } x_i \text{ is even and } x_i < d_i \\ x_i & \text{otherwise} \end{cases}$$

Once EA visits the BV node (and is destroyed there), the LEA and the SAs become aware of the location of the new BV nodes. These are precisely the unexplored neighbours of the original BV node v and, because of the structure of the traversal, their location is precisely determined knowing the coordinates of v; in fact, the set of unexplored neighbours of $v = (x_1, x_2, ..., x_q)$ is $N_{un}(v) = \{(x_1^+, x_2, ..., x_q), (x_1, x_2^+, ..., x_q), ..., (x_1, x_2, ..., x_q^+)\} \setminus \{v\}$, where

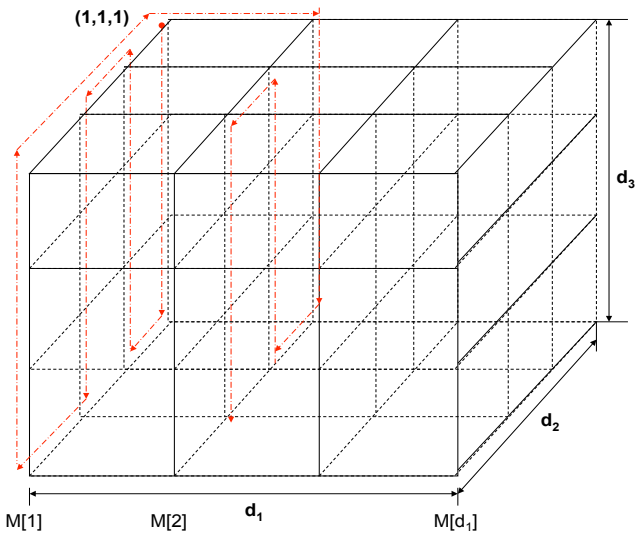

Figure 2: BV exploration in 3D

$$x_i^+ = \begin{cases} x_i + 1 & \text{if } i \leq q-1 \text{ and } x_i < d_i \\ x_i + 1 & \text{if } i = q \text{ and } x_i \text{ is odd and } x_i < d_i \\ x_i - 1 & \text{if } i = q \text{ and } x_i \text{ is even and } x_i > 1 \\ x_i & \text{otherwise} \end{cases}$$

Thus, once v is identified, sequentially, $2q$ agents surround each node $u \in N_{un}(v)$ and an additional agent enters it destroying the BV resident there and the instances that it generates.

Theorem 4.4 *Protocol* BVD-qG*, performs a BV decontamination of a q-dimensional Grid using* $3q+1$ *agents and at most* $q+1$ *casualties.*

Proof: Let $v = (x_1, \ldots, x_q)$ $(1 \leq x_i \leq d_i)$ be the node containing the BV. When EA moves to v, it will be destroyed and the BV will move to all neighbours of v. When this happens, according to the algorithm, all the neighbours in $N_{ex}(v)$ are protected (either by LEA or by a SA) and will not be infected by the BV; this means that the BV can safely move only to the neighbours in $N_{un}(v)$. Since $|N_{un}(v)| \leq q$, there will be at most q new BVs. Since the q-grid is a triangle-free graph, these nodes have no edges in common. This means that these BV nodes can be (sequentially and separately) surrounded and destroyed using at most $3q$ agents (including SAs and LEA): one to enter each BV node, and $2q$ to protect the neighbours. Hence, in addition to EA, at most q agents die, and the total number of employed agents is $3q+1$. □

It is not difficult to verify that the spread and the size of Protocol *BVD-qG* are both *optimal.*

Theorem 4.5 *Let* M *be a q-dimensional grid. Then, regardless of the number of nodes,* $spread(M) = q+1$ *and* $size(M) = 3q+1$.

Proof: Since the very first node visited by any algorithm could be a BV, at least q more BV will be generated; that is, for any solution protocol P, $spread_P(M) \geq q+1$. Of these generated BVs at least one has $2q$ neighbours; furthermore, since M is triangle-free, these neighbours have no edges in common, and none of them is a BV node. This means that at least $2q$ agents are needed in addition to the $q+1$ casualties; that is, for any solution protocol P, $size_P(G) \geq 3q+1$. □

Let us now consider the number of moves performed by the agents.

Theorem 4.6 *A q-dimensional Grid of size $d_1 \times d_2 \ldots \times d_q$ can be decontaminated with at most $O(qn) = O(m)$ moves and $\Theta(n)$ time.*

Proof: Since the length of the traversed path until the BV is found is $O(n)$ in the worst case, the number of moves by LEA, EA and each SA is $O(n)$. Since there are at most q shadowing agents, the total number of moves until the BV is found is $O(qn)$ in the worst case. At that point, there are at most q BV nodes, which are surrounded and eliminated sequentially. Each such node is surrounded by at most $q+1$ agents, each performing $O(1)$ moves, which gives $O(q^2)$ moves in total for surrounding and elimination. Since $q < n$, the bound follows. The total time is obviously $O(n)$ for the first phase, and constant in the second. □

5 BV Decontamination of Tori

A Torus is a regular graph obtained from a grid by adding wrap-around links to the "border" nodes. In a $q-$dimensional torus every node has $2q$ neighbours.

The algorithm to decontaminate the BV in a q-dimensional torus, called *BVD-qT*, follows a strategy very similar to the one used for the q- dimensional Grid described in Section 4.2.

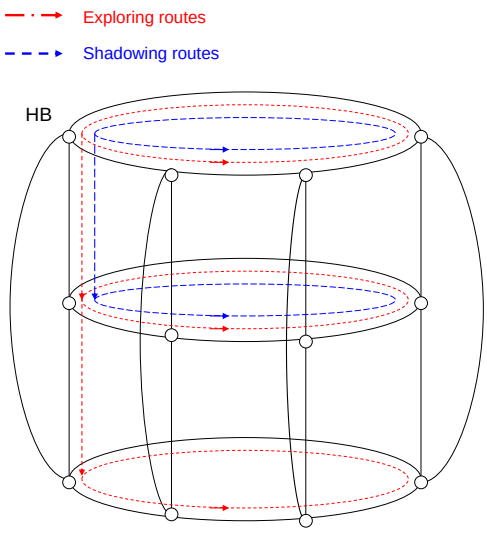

Figure 3: BV exploration in Tori

Let T be a q-dimensional torus of size $d_1 \times \ldots \times d_q$ and let each node of T be denoted by its coordinates $(x_1, \ldots, x_q)$, $1 \leq x_i \leq d_i$. Without loss of generality, let $(1, 1, \ldots, 1)$ be the homebase. Informally, the multi-dimensional torus is partitioned into $d_1 \times \ldots \times d_{q-2} \times d_{q-1}$ *rings* of size d_q. The exploration procedure traverses a ring and, when back to the starting point, proceeds to another ring, with a neighbouring starting point. In Figure 3 the traversal is shown for $q = 2$.

Note that, because of the lack of borders, the spread of the BV might be higher in Tori than in Grids. For example, if one of the nodes visited during the first traversed ring is the BV, the BV will reach all its neighbours except the one occupied by LEA, inducing a casualty count of $4q$.

The Shadowing procedure is identical to the one described for the Grid: when node v is to be explored, all nodes in $N_{ex}(v)$ are occupied by a SA to protect them. Once the BV (say node v) is visited by EA, each node of $N_{un}(v)$ is surrounded sequentially and then eliminated like it is done for the Grid.

Theorem 5.1 *Protocol* BVD-qT*, performs a BV decontamination of a q-dimensional Grid using $4q$ agents with $2q$ casualties with at most $O(qn)$ moves and $\Theta(n)$ time.*

Proof: Let $N_{ex}(v)$ (resp. $N_{un}(v)$) denote the set of explored (resp. unexplored) neighbours of node v. The number of casualties is equal to $1 + |N_{un}(v)| \leq 2q$. To surround a BV node it always takes $2q$ agents, while for the shadowing $|N_{ex}(v)| = 2q - |N_{un}(v)|$ are deployed. Thus, in addition to the agents that die in the BVs, at most $\max\{2q, 2q - |N_{un}|(v)\} = 2q$ agents are used for shadowing and surrounding. The total number of agents employed by the algorithm is then: $1 + |N_{un}| + 2q \leq 4q$.

It is easy to see that the asymptotic number of moves as well as the time are the same as the ones determined for the q-Grid. □

It is not difficult to verify that both the spread and the size of Protocol *BVD-qT* are optimal.

Theorem 5.2 *Let T be a q-dimensional torus. Then, regardless of the number of nodes, $spread(T) = 2q$ and $size(T) = 4q$.*

Proof: Since a q-dimensional torus is a triangle-free $2q$-regular graph, the optimality follows directly from Properties 3.6 and 3.7. □

6 BV Decontamination of Hypercubes

6.1 The Hypercube and its Properties

The hypercube is a classical topology for interconnection networks. A 1-dimensional hypercube is simply composed by two connected nodes, one labeled 0 and the other labeled 1. A q-dimensional hypercube can be constructed recursively by two copies of the (q-1)-dimensional hypercubes with links between each pair of corresponding nodes (e.g. nodes with the same labels) in the two sub-cubes. The name of each node in one of the sub-cubes is changed by prefixing it with a bit 0, and the name of the corresponding node in the other is changed by prefixing it with bit 1. A q-dimensional hypercube (q-Hypercube) has 2^q nodes, and $m = n \cdot \frac{q}{2} = \frac{1}{2} n \log_2 n$ links. Associating a q-bit string to each node as described above, an edge between two nodes u and v exists if and only if u and v differ in exactly one bit position (which is called the "dimension" of the edge).

The hypercube has several interesting properties that make it a desirable topology in many applications. The hypercube clearly admits an Hamiltonian tour. A particular Hamiltonian tour is given by following the so called *Gray code*, and it is described below.

The Gray code (also called reflected binary code) is a binary numeral system where two successive values differ in only one bit. The binary-reflected Gray code list G_q for q bits can be constructed recursively from list G_{q-1} by reflecting the list (i.e. listing the entries in reverse order), concatenating the original list with the reversed list, prefixing the entries in the original list with a binary 0, and then prefixing the entries in the reflected list with a binary 1. Let $0G_i$ indicate list G_i where every element is prefixed by 0 (resp. 1), let $\overline{G_i}$ indicate list G_i reversed, and let $\circ$ indicate concatenation of lists. We then have: $G_q = 0G_{q-1} \circ 1\overline{G_{q-1}}$. For example $G_1 = [0, 1]$, $G_2 = [00, 01, 11, 10]$, $G_3 = [000, 001, 011, 010, 110, 111, 101, 100]$, and so on.

The Gray code provides a way to traverse a hypercube starting from node $(00\ldots0)$: each and every node is visited once and a sub-cube is fully visited before proceeding to the next.

The properties of the Gray code allows one to determine easily the position $Pos(u)$ of a node $u = (d_1 \ldots d_q)$ in the traversal of the hypercube following the Gray code. In fact, we have that $Pos(u) = (d'_1 \ldots d'_q)$ where $d'_1 = d_1$ and, for $1 < i \leq q$:

$$d'_i = \begin{cases} d_i & \text{if } d_{i-1} = 1 \\ 1 - d_i & \text{if } d_{i-1} = 0 \end{cases}$$

In a similar way, given a binary number $b = (b_1, \ldots b_q)$, the the b-th node $G(b)$ in the traversal of a q-hypercube following the Gray code (starting from 0) is $G(b) = (b'_1, \ldots b'_q)$, where $b'_1 = b_1$ and, for $1 < i \leq q$:

$$b'_i = \begin{cases} b_i & \text{if } b_{i-1} = 0 \\ 1 - b_i & \text{if } b_{i-1} = 1 \end{cases}$$

Particularly useful are the functions $Pred(u)$ and $Succ(u)$ that, given a node $u = (d_1, \ldots d_q)$ determine all its successors and all its precedessors in the Gray code traversal of the hypercube. Those functions are defined below, and they can clearly be computed locally by LEA: $Pred(u) = \{G_q(i) : i < Pos(u)\}$ and $Succ(u) = \{G_q(i) : i > Pos(u)\}$.

Another useful function which is locally computable is $Next(u)$ which, given a node u, returns the next node in the Gray code traversal: $Next(u) = G(Pos(u) + 1)$.

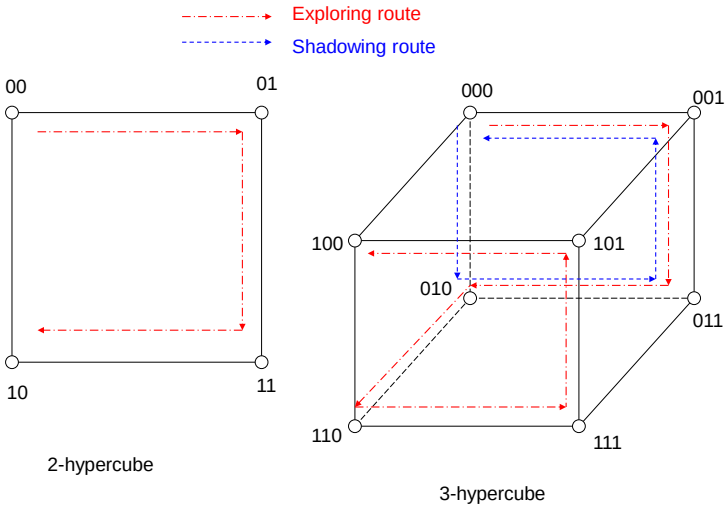

Figure 4: BV exploration in 2 and 3-Hypercube

6.2 Decontaminating a BV in q-Hypercube

Also in the case of the hypercube, the agents first perform a shadowed exploration, and then proceed to the elimination once the BV is detected and triggered. In the shadowed exploration, the two exploring agents EA and LEA traverse the hypercube following the Gray code, with the shadowing agents protecting at each step the nodes in the explored area adjacent to the node under exploration. EA goes ahead first, leaving LEA in the previously explored node, and if the BV arrives instead of EA, LEA realizes that the BV has been hit. In the particular case of a q-hypercube, hitting the BV triggers its propagation to the q neighbours, some of which are already explored (and protected by the SAs), while some are not explored yet. Those neighbours protected by the SAs will not be infected by the arrival of the new BV, the others will become BV nodes; they will then be surrounded and eliminated (procedure Elimination and Surrounding).

Example: 3-Hypercube. As an example of shadowed exploration, let us consider in detail the case of the 3-Hypercube. The exploring agents follow the Gray code traversal, which in this case consists of the sequence of nodes: $000, 001, 011, 010, 110, 111, 101, 100$ and is depicted in Figure 4. Two shadowing agents (a and b) are needed overall, and they also perform a traversal of a subcube starting from 000. More precisely, no shadow is required while EA is visiting 001 and 011; when EA explores node 010, both a and b are still in node 000 to protect it, no shadow is needed when EA is exploring 110 (because both neighbours different from the predecessor are unexplored), the shadowing agents then proceed to 010 when EA visits 110, then one of the shadowing agents (say a) stays here, and b moves to 011 to protect it while EA visits 111, b then moves to 001 when EA is exploring 101, and finally b moves to 000 and a moves to 110 when EA is exploring 100. Note that synchronicity of the system allows the shadowing agents to correctly synchronize their movements with the exploration agents. Also note that one of the shadowing agents (a in the example) initially follow the other shadowing agents without being needed ("inactive"). Shadowing agent a becomes

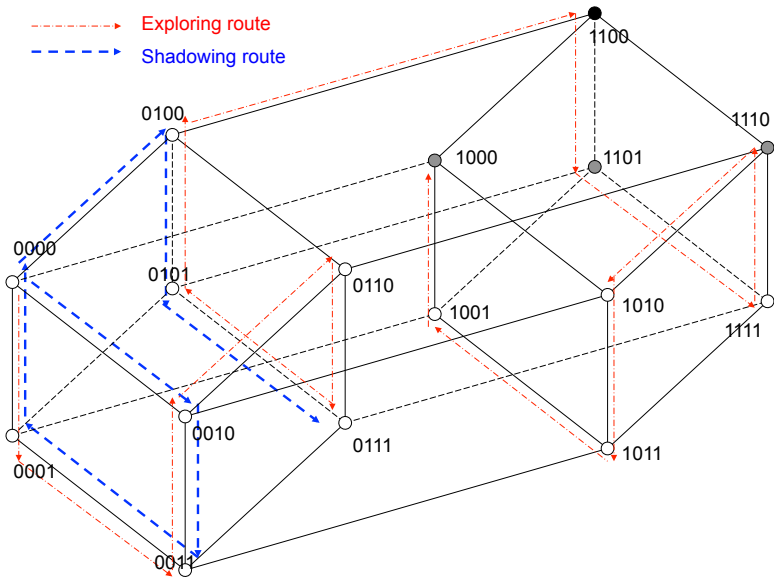

Figure 5: BV exploration in 4-Hypercube

needed ("active") when the exploration reaches node 100. In other words, in the 3-Hypercube one shadowing agent performs the reverse of the first half of the grey code, while the second shadow moves to the node connecting the two 2-Hypercubes to protect from the exploration of the last node.

In general, in the case of a q-Hypercube, one step in the shadowed exploration, with EA and LEA both at a safe node u, is described by the following algorithm:

ALGORITHM *BVD-qH*: SHADOWED EXPLORATION

Agents EA and LEA are at safe node u.
1. Agents compute $Next(u)$:
   $Next(u) = G(Pos(u) + 1)$. Let $u' = Next(u)$
2. Shadowing agents compute $N_{ex}(u') = N(u') \cap Pred(u')$
   Shadowing agents move to occupy $N_{ex}(u') \setminus u$
3. EA moves to $Next(u)$.
4. If unarmed, EA returns to u and both EA and LEA move to $Next(u)$.

When moving to occupy their computed positions, the paths followed by the $q-1$ shadowing agents can be intuitively described recursively as follows:

Shadow(Q, q)

- 1. For $q = 3$, the paths of the 2 shadowing agents are as described in the example above.
- 2. For $q > 3$: the q-Hypercube Q is decomposed along dimension q in the two $(q-1)$-subcubes Q^1 and Q^2
  - 2.1. During the exploration of Q^1, $q-2$ shadowing agents follow the paths indicated by **Shadow**$(Q^1, q-1)$, while one is inactive.
  - 2.2. During the exploration of Q^2, the $q-2$ shadowing agents already employed for Q^1 now follow the paths indicated by **Shadow**$(Q^2, q-1)$.
    The extra shadowing agent covers the path in Q^1 that corresponds to the exploration path of EA in Q^2.

As an example, the shadowed exploration routes in the 4-Hypercube by the 3 shadowing agents are shown in Figure 5. **figures to be re-done**. Indicated are also the timing constraints: label t on a SA movement has to be completed before the movement labeled t by EA takes place.

Notice that the starting point in each subcube depends on the exploration path followed by LEA, that is on the Gray code.

The shadowed exploration continues until EA meets the BV, at which time EA dies, LEA receives a new virus from the node v just explored by EA detecting the virus, and the elimination phase starts. At this point LEA, together with the shadowing agents SAs fully cover $N_{ex}(v)$, and new BVs have spread to $N_{un}(v)$.

ALGORITHM *BVD-qH*: SURROUNDING AND ELIMINATION

BV originally in v now spread to $N_{un}(v) = N(v) \cap Succ(v)$
LEA and SAs are at the nodes $N_{ex}(v)$

1. LEA moves to v.
2. For each $u \in N_{un}(v)$:
   2.1 Agents move to each unoccupied $z \in \{N(u)\}$
   2.2 An agent moves to u to permanently remove that BV.

6.3 Complexity analysis

The overall number of agents needed for the whole process depends on the position of the BV with respect to the traversal.

Theorem 6.1 *Protocol* BVD-qH*, performs a BV decontamination of a q-Hypercube using* $2q$ *agents and* q *casualties.*

Proof: One agent dies in the BV. Let v be the node containing the BV. $|N_{ex}(v)|$ agents are first employed to shadowing and then reused for surrounding and eliminating. Since the neighbours of the BV are disjoint, nodes in $N_{un}(v)$ are surrounded and eliminated sequentially. There are $q-|N_{ex}(v)|$ such nodes and for each of them, q agents are needed for surrounding and 1 for the elimination. Overall we then need $q-|N_{ex}(v)|+1$ agents for elimination (these agents die and cannot be reused), and q for surrounding sequentially each of the nodes in $N_{un}(v)$. The total number of agents employed is then $2q+1-|N_{ex}(v)|$. Since $|N_{ex}(v)| \geq 1$, the bound follows. □

It is not difficult to verify that both the spread and the size of Protocol $BVD\text{-}qH$ are optimal.

Theorem 6.2 *Let* H *be a q-Hypercube, we have that* $spread(H) = q$ *and* $size(H) = 2q$.

Proof: Since a q-Hypercube is a triangle free q-regular graph, the optimality follows directly from Properties 3.6 and 3.7. □

Let us now consider the number of moves.

Theorem 6.3 *Protocol* BVD-qH*, performs a BV decontamination of a q-Hypercube with at most* $O(n \log n)$ *moves and time* $\Theta(n)$.

Proof: First note that the number of moves performed by the exploring agents is $4(Pos(v)-1)+1$, where v is the node containing the BV. So we have a total of at most $4n$ moves for the exploring agents. The number of moves performed by each shadowing agent is bounded by n and the total number of shadowing agents employed is $q-1 = \log n - 1$. We then have that the total number of movements by the shadowing agents is $O(n \log n)$.

Finally, let us now consider the number of moves performed for surrounding and eliminating the new BVs. There are q such new BVs, to be treated sequentially. For each of them there are q neighbours to be occupied by the q agents. To reach any of these assigned nodes, an agent performs a constant number of moves. So, each new BV is surrounded and eliminated in at most $O(q)$ moves

and all the BVs are eliminated in at most $O(q^2) = O(\log^2 n)$ moves. Adding all these costs, the move complexity is then $O(n \log n)$.

As for the time complexity. The time required for the exploration phase is equal to the number of moves of EA, which is $O(n)$; the time required for the surrounding and elimination phase is constant. □

7 Conclusions and Open Problems

7.1 Conclusions

In this paper we have introduced the problem of decontaminating a network from a *black virus*, a process harmful to both sites and agents and capable of reactive mobility. This problem presents aspects that are not covered by the classical problem studied in the literature: black hole search and intruder capture. In fact a black virus is capable to harm agents (like a black virus but unlike an intruder) and is capable to spread (like an intruder but unlike a black hole). As such, it opens a new line of investigation on security threats in networked systems supporting mobile agents.

We have started this investigation defining the model and establishing basic properties. We have then examined in detail the problem for three classic classes of interconnection networks: grids, tori, and hypercubes. We have presented decontamination protocols for each class (for all dimensions), and analyzed their complexity in terms of spread of the virus and total number of agents, and we have shown that our protocols are optimal in both measures. Although described for a synchronous setting, the solutions easily adapt to asynchronous ones (requiring only coordination among the shadowing and the exploring agents), all the results hold also in this case, and the extra cost consists of an additional $O(n)$ moves in total for coordinating the activities. An advantage of our solutions is that the agents use only local information to execute the protocol.

7.2 Open Problems

The results of this paper open many research problems and pose new questions.

An immediate open problem is to study the black virus decontamination problem in other classes of networks. Since the classes studied here are all of regular graphs, a particular focus should be on *irrregular* networks; so far, our knowledge is limited to *trees* (see Lemma 3.1).

A related important research question is how to decontaminate *arbitrary* networks; i.e., to design a "generic" protocol that allows to decontaminate every network. We have already started to investigate some of these problems.

Another interesting research direction is to investigate the role of *topological knowledge*; e.g., to study the impact that presence or absence of a network map has on the complexity of decontamination; to discover the relationship between topological parameters and complexity; etc.

This paper is a starting point for a better understanding of the impact that the severity of the security threat (i.e., the power of the harmful entity) has on the complexity of the defence mechanism (i.e. the complexity of the solution protocol). For example, although a black virus combines some harmful properties of a black hole (as defined by the black hole search problem) with some of an intruder (as defined by the intruder capture problem), a black virus it is not all powerful.

In particular, a black virus remains hidden until it is triggered when a specific condition (the arrival of a system agent) occurs, much like a *logic bomb*; that is, its behaviour, including its mobility, is *reactive*. Once triggered, it copies itself and spreads to neighbouring computers. Of clear interest is the study of the more complex problem decontamination when the behaviour of the Black Virus is *proactive*, like a pure virus, when the spreading is spontaneous, that is not triggered by specific events.

Currently the power of the black virus is restricted also because of *limited spreading* (BV moves only to the immediate neighbourhood). An important research direction is to investigate which of these restrictions (reactivity and limited spreading) is the most severe, which can be reduced or lifted and the problem still be solvable, etc.; in other words, to determine whether the system is

capable of sustaining a more powerful threat, how much more power the agents would need to have to be able to sustain a stronger threat, etc.

Acknowledgments. The authors would like to thank the anonymous referees whose comments have helped improving the clarity of the paper. This work has been supported in part by NSERC Discovery Grants, and by Prof. Flocchini's University Research Chair.

References

[1] B. Balamohan, S. Dobrev, P. Flocchini, and N. Santoro. Asynchronous exploration of an unknown anonymous dangerous graph with O(1) pebbles. *Proceedings of 19th International Colloquium on Structural Information and Communication Complexity* (SIROCCO), 279-290, 2012.

[2] B. Balamohan, P. Flocchini, A. Miri, and N. Santoro. Time optimal algorithms for black hole search in rings. *Discrete Mathematics, Algorithms and Applications*, 3(4): 1-15, 201

[3] L. Barrière, P. Flocchini, F.V. Fomin, P. Fraignaud, N. Nisse, N. Santoro, and D.M. Thilikos. Connected graph searching. *Information and Computation*, 219: 1-16, 2012.

[4] L. Barrière, P. Flocchini, P. Fraignaud, and N. Santoro. Capture of an intruder by mobile agents. *Proceedings of 14th Symposium on Parallel Algorithms and Architectures* (SPAA), 200-209, 2002.

[5] L. Blin, P. Fraignaud, N. Nisse, and S. Vial. Distributed chasing of network intruders. *Theoretical Computer Science* 399 (1-2): 12-37. 2008.

[6] J. Chalopin, S. Das, and N. Santoro. Rendezvous of mobile agents in unknown graphs with faulty links. *Proceedings of 21st International Symposium on Distributed Computing (DISC)*, pages 108–122, 2007.

[7] J. Chalopin, S. Das, A. Labourel, and E. Markou. Tight bounds for scattered black hole search in a ring. *Proceedings of 18th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 186–197, 2011.

[8] J. Chalopin, S. Das, A. Labourel, and E. Markou. Black hole search with finite automata scattered in a synchronous torus. *Proceedings of 25th International Symposium on Distributed Computing (DISC)*, pages 432–446, 2011.

[9] C. Cooper, R. Klasing, and T. Radzik. Searching for black-hole faults in a network using multiple agents. *Proceedings of 10th International Conference on Principles of Distributed Systems* (OPODIS), pages 320–332, 2006.

[10] C. Cooper, R. Klasing, and T. Radzik. Locating and repairing faults in a network with mobile agents. *Theoretical Computer Science* 411(14–15):1638–1647, 2010

[11] J. Czyzowicz, S. Dobrev, R. Královic, S. Miklík, and D. Pardubská. Black hole search in directed graphs. *Proceedings of 16th International Colloquium on Structural Information and Communication Complexity* (SIROCCO), pages 182–194, 2009.

[12] J. Czyzowicz, D. R. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71(2–3):229–242, 2006.

[13] J. Czyzowicz, D. R. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability & Computing*, 16(4):595–619, 2007.

[14] Y. Daadaa, P. Flocchini, N. Zaguia. Network decontamination with temporal immunity by cellular automata. *Proceedings of the 9th International Conference on Cellular Automata for Research and Industry* (ACRI), 287-299, 2010.

[15] M. D'Emidio D. Frigioni, and A. Navarra. Exploring and making safe dangerous networks using mobile entities. *Proceedings of 12th International Conference on Ad Hoc Networks and Wireless* (ADHOC-NOW), pp. 136-147, 2013.

[16] D. Dereniowski. Connected searching of weighted trees. *Theoretical Computer Science* 412 (41): 5700-5713, 2011.

[17] S. Dobrev, P. Flocchini, R. Královic, P. Ruzicka, G. Prencipe, and N. Santoro. Black hole search in common interconnection networks. *Networks*, 47(2):61–71, 2006.

[18] S. Dobrev, P. Flocchini, R. Královic, and N. Santoro. Exploring an unknown dangerous graph using tokens. *Theoretical Computer Science* 472: 28-45, 2013.

[19] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. *Distributed Computing*, 19(1):1–19, 2006.

[20] S. Dobrev, P. Flocchini, and N. Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48:67–90, 2007.

[21] S. Dobrev, N. Santoro, and W. Shi. Using scattered mobile agents to locate a black hole in a unoriented ring with tokens. *International Journal of Foundations of Computer Science*, 19 (6), 1355-1372, 2008.

[22] P. Flocchini, M.J. Huang, and F.L. Luccio. Decontamination of hypercubes by mobile agents. *Networks* 52 (3): 167-178, 2008.

[23] P. Flocchini, M.J. Huang, and F.L. Luccio. Decontaminating chordal rings and tori using mobile agents. *International Journal on Foundations of Computer Science* 18 (3): 547-563, 2006.

[24] P. Flocchini, D. Ilcinkas, and N. Santoro. Ping pong in dangerous graphs: optimal black hole search with pebbles. *Algorithmica*, 62(3-4): 1006–1033, 2012.

[25] P. Flocchini, M. Kellett, P. Mason, and N. Santoro. Searching for black holes in subways. *Theory of Computing Systems*, 50(1): 158–184, 2012

[26] P. Flocchini, M. Kellett, P. Mason, and N. Santoro. Map construction and exploration by mobile agents scattered in a dangerous network. *Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium* (IPDPS), pages 1–10, 2009.

[27] P. Flocchini, F. Luccio, L. Pagli, N. Santoro. Optimal network decontamination with threshold immunity. *Proceedings of 8th International Conference on Algorithms and Complexity* (CIAC), pages 234–245, 2013.

[28] P. Flocchini, F.L. Luccio, and L.X. Song. Size optimal strategies for capturing an intruder in mesh networks. *Proceedings of International Conference on Communications in Computing* (CIC), pages 200-206, 2005.

[29] P. Flocchini, B. Mans, and N. Santoro. Tree decontamination with temporary immunity. *Proceedings of 19th International Symposium on Algorithms and Computation* (ISAAC), 330–341, 2008.

[30] P. Flocchini and N. Santoro. *Distributed Security Algorithms For Mobile Agents.* In J. Cao and S.K. Das, editors, *Mobile Agents in Networking and Distributed Computing*, chapter 3. Wiley, 2011.

[31] F.V. Fomin, D.M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science* 399(3): 236-245, 2008

[32] F.V. Fomin, D.M. Thilikos, and I. Todineau. Connected graph searching in outerplanar graphs. *Proceedings of 7th International Conference on Graph Theory* (ICGT), 2005.

[33] P. Glaus. Locating a black hole without the knowledge of incoming link. *Proceedings of 5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks* (ALGOSENSOR), pages 128–138, 2009.

[34] D. Ilcinkas, N. Nisse and D. Soguet. The cost of monotonicity in distributed graph searching. *Distributed Computing* 22 (2), 117-127, 2009.

[35] N. Imani, H. Sarbazi-Azadb, and A.Y. Zomaya. Capturing an intruder in product networks. *Journal of Parallel and Distributed Computing* 67 (9): 1018–1028, 2007.

[36] N. Imani, H. Sarbazi-Azad, A.Y. Zomaya, and P. Moinzadeh. Detecting threats in star graphs. *IEEE Transactions on Parallel and Distributed Systems* 20 (4): 474-483 , 2009.

[37] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Theoretical Computer Science*, 384(2-3):201–221, 2007.

[38] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Approximation bounds for black hole search problems. *Networks*, 52(4):216–226, 2008.

[39] A. Kosowski, A. Navarra, and C. M. Pinotti. Synchronous black hole search in directed graphs. *Theoretical Computer Science*, 412(41):5752 – 5759, 2011.

[40] F. Luccio and L. Pagli. A general approach to toroidal mesh decontamination with local immunity. *Proceedings of 23rd IEEE International Parallel and Distributed Processing Symposium* (IPDPS), 1-8, 2009.

[41] F. Luccio, L. Pagli, and N. Santoro. Network decontamination in presence of local immunity. *International Journal of Foundation of Computer Science* 18(3): 457–474, 2007.

[42] F.L. Luccio Contiguous search problem in Sierpinski graphs. *Theory of Computing Systems* 44(2): 186-204, 2009.

[43] N. Nisse. Connected graph searching in chordal graphs. *Discrete Applied Mathematics* 157 (12): 2603-2610, 2009

[44] P. Shareghi, H. Sarbazi-Azad, and N. Imani. Capturing an intruder in the pyramid. *Proceedings of International Computer Science Symposium in Russia* (CSR), 580-590, 2006.

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

[46] B. Yanga, D. Dyerb, and B. Alspach. Sweeping graphs with large clique number. *Discrete Mathematics* 309 (18): 5770-5780, 2009.

APPENDIX

A Exploration and Shadowing

In all proposed algorithms, regardless of the network topology, the execution, until the BV is found, consists of a sequence of *exploration steps*. Each step comprises of an ordered sequence of operations:

1. At the beginning of each step, EA and LEA are at a safe node v.

2. All agents determine the new node $u = next(v)$ to be explored among the neighbours of v as well as the set $N_{ex}(u) = \{u_0, u_1, ..., u_q\}$ of the already explored neighbours of u, where $u_0 = v$.

3. Let $\{S_1, S_2, ..., S_p\}$ be the shadowing agents; note that, by construction, in each of the examined classes of networks, $p \geq q$. Each $u_i \in N_{ex}(u)$ is assigned to a SA S_i, $1 \leq i \leq q$; $u_0 = v$ is assigned to LEA and to $S_{q+1}, ..., S_p$. The agents move to the assigned nodes

4. EA moves to $next(v)$.

Note that the assignment of nodes to SAs can be calculated according to optimization criteria (e.g., u_i is vertex in $N_{ex}(u)$ closest to S_i, where ties are broken using identities) so to minimize the number of moves. Observe that the movements of each S_i over time defines a specific trajectory, e.g., a snake-like path in a grid; see Figure 5 for an example in a 4-dimensional hypercube.

How the proper sequencing of these operations is ensured depends on whether or not the system is synchronous.

Synchronous Agents

Assume: it takes one time unit for each movement (by agent or BV); computing and processing is negligible (instantaneous)

Let t_j denote the time when the j-th step starts, and let $\hat{t}_j$ denote the time when EA moves to $next(v)$ in that step. Initially (i.e. at time t_1) all agents are at the homebase, and know the position of every other agent; all the algorithms maintain the invariant that, at time t_j, all agents know the position of every other agent and that the j-th step is starting.

Let d_i be the distance between the current position of s_i at time t_j and the assigned node u_i, and let $d = Max\{d_i\}$. This means that, by the time $t_j + d$ all nodes have reached their position for this step. Hence EA moves to $u = next(v)$ at time $\hat{t}_j = t_j + d$, when all already explored neighbours of u are protected by agents. By time $\hat{t}_j + 2 = t_j + d + 2$ all agents know whether the BV has been detected: a BV clone will arrive at the locations of the agents at that time if and only if the BV was at u. If the BV is not detected, step $j + 1$ can start at time $t_{j+1} = \hat{t}_j + 3 = t_j + d + 3$ with LEA moving to u; notice that all agents know both t_{j+1} and the position of every other agent at that time. If the BV is detected, the second phase of the protocols is started.

Asynchronous Agents

It takes a finite but unpredictable amount of time to perform any operation. Communication occurs when two agents meet.

In an *asynchronous* system, the LEA will act as overall coordinator, responsible for the communication with and coordination of EA and the shadowing agents.

Let t_j denote the time when the j-th step starts. Initially (i.e. at time t_1) all agents are at the homebase; all the algorithms maintain the invariant that, at time t_j, LEA knows the position of all the other agents and that the j-th step is starting.

Starting at time t_j, LEA moves sequentially to notify all SAs of the new step. To notify s_i ($1 \leq i \leq p$) LEA moves to the position currently occupied by s_i; it communicates to s_i its assigned destination node u_i (where $u_l = v$ for $l > q$); both s_i and LEA move (asynchronously and independently) to u_i; when LEA arrives at u_i, it waits until u_i arrives. Once all SA have been notified and have moved to the assigned positions, LEA returns to v, and notifies EA to move to u.

If BV is not at u, upon arriving at u, EA returns at v and notifies LEA that a new step must be performed; both LEA and EA move to u; the time when they both arrive is called t_{j+1} and the $(j+1)$-th step starts.

If BV is at u, when EA arrives there, it is destroyed and it causes BV to move to all neighbours of u, including v where LEA is waiting. When this occurs, the second phase of the protocol starts. Also in that phase, LEA will coordinate the movements of the agents.

B Surrounding and Elimination

Once the BV has been detected and has moved to the unexplored neighbours of its original location, in all proposed algorithms the surviving agents start the second phase. In this phase, all BV nodes are surrounded and the permanent elimination of the BVs is performed.

Let u be the original location of BV, and let $N_{un}(u) = \{z_1, ..., z_k\}$ denote the set of the unexplored neighbours of u when EA enters u during the exploration. The second phase consists of a sequence of k *elimination steps* to permanently remove the BV from $N_{un}(u)$. In the j-th step, the following operations are performed:

1. Let $M(z_j) = N(z_j) \setminus N_{un}(u) = \{v_1, ..., v_{q_j}\}$ be the set of non-BV neighbours of z_j, and let $\{A_1, A_2, ..., A_{p_j}\}$ be the surviving agents at the beginning of stage j. To each agent A_i $(1 \leq i \leq p_j)$, is assigned node $v_i \in M(z_j)$, where $v_l = v_1$ for $l > q_j$. Note that by construction, $p_j > q_j$; hence more than one agent is assigned to v_1.

2. The agents move to the assigned nodes.

3. A_1 moves to z_j to permanently remove the BV from there.

Note that the assignment of nodes to agents can be calculated according to optimization criteria so to minimize the number of moves; e.g., v_i is vertex in $M(z_j)$ closest to v_i, where ties are broken using identities $(1 \leq i \leq q_j)$.

How the proper timing of these operations is ensured depends on whether or not the system is synchronous.

Synchronous Agents

Let t_1 the time when the BV is detected; notice that, by construction, all agents (except EA that is destryed by BV) will detect it at the same time. At that time the first step of the second phase starts. Let t_j denote the time when the j-th step starts $(1 \leq j \leq k)$, and let $\hat{t}_j$ denote the time when A_1 moves to z_j in that step; all the algorithms maintain the invariant that, at time t_j, all agents know the position of every other agent and that the j-th step is starting.

Let w_i be the distance between the current position of A_i at time t_j and the assigned node v_i, and let $w = Max\{w_i\}$. This means that, at time $t_j + w$ all nodes have reached their position for this step.

Agent A_1 moves to z_j at time $\hat{t}_j = t_j + w$, when all non-BV neighbours of z_j are protected by agents. At time $\hat{t}_j + 2 = t_j + w + 2$ a BV clone will arrive at the locations of the agents. Step $j+1$ can start at time $t_{j+1} = \hat{t}_j + 3 = t_j + w + 3$; notice that all agents know both t_{j+1} and the position of every other agent at that time, preserving the invariance.

Asynchronous Agents

In an *asynchronous* system, in addition to the synchronization and coordination of the agents, there is the additional difficulty of dealing with the unpredictable delay before a clone BV arrives at a node protected by an agent. This problem occurs immediately upon the arrival of EA to the original location u of the BV. Let W_j denote the set of nodes where there are agents at the beginning of step j. By definition, $W_j = M(z_{j-1})$ for $j > 1$; at the beginning of the first step, W_1 is instead the set of the locations of all agents when EA moves to u.

The coordinator role is played by LEA (called A_{p_j} in step j). The j-th step starts at time t_j when the BV clone arrives at the node where LEA is waiting, detecting the end of step $j-1$ (if $j > 1$) or of the first phase ($j = 1$). When this occurs LEA does as following.

First LEA moves sequentially to all nodes in W_j. In each of this nodes, it waits until the BV clone has arrived; it then communicates the assigned destination node(s) to the agent(s) resident there.

Then LEA moves sequentially to the nodes in $M(z_j)$ (in reversed order, ending in v_1). In each of this nodes, it waits until all the agents assigned to that node have arrived.

Finally, LEA notifies A_1 to move to z_j. At this point, LEA waits until the BV clone arrives, determining the end of the j-th step.