Capturing an Intruder in the Hypercube by Mobile Agents *

Paola Flocchini and Miao Jun Huang
School of Information Technology and Engineering
University of Ottawa, 800 King Eduard,
Ottawa, Ontario, K1N 6N5, Canada
{flocchin,mhuang}@site.uottawa.ca

Flaminia L. Luccio†
Dipartimento di Matematica e Informatica
University of Trieste
via Valerio 12, 34100, Trieste, Italy
luccio@dsm.univ.trieste.it

Abstract

In this paper we consider the problem of searching for an intruder in a network. There is a team of collaborative software agents that are deployed to capture a hostile intruder (e.g., a virus). These agents move along the network links and the intruder has the capability of escaping arbitrarily fast. We propose four different strategies for the solution of the problem in a widely studied topology: the hypercube network. In the first strategy one of the agents acts as a coordinator forcing the other agents to move in a precise way; this strategy employs an optimal number of agents ($\Theta(\frac{n}{\sqrt{\log n}})$), with $O(n \log n)$ moves, and runs in $O(n \log n)$ time steps. The second strategy is devised for a model where the agents are allowed to "see" the state of their neighbours. In this case, the computation is local, i.e., there is no need of a coordinator and agents can move autonomously. In this setting the solution requires $\frac{n}{2}$ agents, but the time complexity is optimal ($\log n$ time steps), and requires the same number of moves ($O(n \log n)$). Finally, the last two strategies are devised for models that assume agents have cloning capabilities (i.e., can create a copy of themselves) and either can "see" the state of their neighbours or move in a synchronous setting. In both cases the bound on the number of moves becomes optimal decreasing to $n - 1$.

Keywords: Graph search, hypercube, mobile agents, intruder capture.

---

*Work partially supported by NSERC and by Abstract Interpretation Design and Applications, MIUR COFIN Project.

†Correspondence may be sent to Flaminia L. Luccio, c/o Dipartimento di Matematica e Informatica, Università degli Studi di Trieste, via Valerio 12/1, 34127 Trieste, phone: 0039/040-5582610, fax: 0039/040-5582623.

1 Introduction

1.1 The Problem

We consider a networked environment where nodes are hosts and links represent connections between hosts. A possibly dangerous piece of software (e.g., a virus) is moving in the network from host to host; we will call such an element, the *intruder*. We assume we have a team of collaborative software agents that can be deployed to protect the network and thus have the goal of localizing and neutralizing the intruder.

The agents are initially located at the same node (the *homebase*), they are autonomous in the sense that each agent has some local memory ($O(\log n)$ bits, where n is the size of the network, suffice for all our algorithms), and can perform local computation. Each agent can move from node to neighbouring node.

In the paper we consider both the cases of asynchronous and synchronous environments. In the *asynchronous* environment every action the agents perform (i.e., computing, moving), takes a finite but otherwise unpredictable amount of time; in the *synchronous* environment we assume that local computations are instantaneous, and it takes one unit of time (one step) to move from node to neighbouring node.

The intruder is also a software agent. To concentrate on the worst case scenario, we assume that given any strategy, the intruder always behaves in the most powerful way. In other words, the intruder moves as if it can "see" the whereabouts of the team of agents, thus avoiding them as much as possible. The intruder is detected when an agent resides on the same node.

The network we consider is the hypercube, which we assume to be labeled according to the classical dimensional labeling. We also assume that the agents know the topology. To each node of the hypercube is associated a local storage area (here called whiteboard) where information about the node and its outgoing links are stored. The whiteboard of a node can be accessed (for reading and writing) by an agents when it resides at the node[1].

Our goal is to devise an efficient strategy for deploying the agents in the network so to localize and neutralize the intruder. Efficiency will be measured in terms of number of agents to be involved, traffic (i.e., number of moves the agents have to perform), and time (or steps)[2].

1.2 Related Works

The intruder capturing problem has been extensively studied in the literature under the name of *graph search* or *decontamination problem* (e.g., see [5, 8, 9, 10, 13]). The graph search problem consists of having a system of tunnels represented by edges of a graph. These tunnels are initially all *contaminated*, and have to be decontaminated (or cleaned) by a sequence of actions executed by a (minimal) set of *searchers*. The problem was first introduced in [11, 12], and there are several variations of it. In particular, in the *node search problem* an edge (x, y) is decontaminated when two searchers are placed, one on x and the other on y, while in the *edge search problem* an edge

[1]Access to the whiteboard is assumed to occur in mutual exclusion.

[2]In the case of asynchronous systems, we will measure ideal time, i.e., assuming that it takes one unit of time, for an agent to traverse a link.

has to be also traversed to be decontaminated. The aim of both problems is to find a sequence of actions for the searchers that permits to reach a state in which all the edges are simultaneously decontaminated, where an action is one of the following operations: (1) place a searcher on a node, (2) remove a searcher from a node. In the edge search problem a third operation is allowed: (3) move a searcher along an edge. The graph search problem has been extensively studied for many classes of graphs, and determining the optimal number of searchers (called *search number*) has been proved to be NP-complete in most known topologies.

It is easy to see that graph search and intruder detection are equivalent problems. If the intruder is allowed to "hide" only on nodes, we are dealing with a node search problem; if the intruder can "hide" also on the edges, then we have an edge search problem. We say that a node is *clean* if either it contains an agent, or all its adjacent nodes are clean, then in the node search problem, the goal is to have all the nodes simultaneously clean. In the rest of the paper we will mostly use the terminology of graph search.

In most graph search variations studied in the literature the results are heavily based on the assumption that a searcher can be initially placed on an arbitrary node and can be arbitrarily moved to any other node. The main difference in our setting is that the agents *cannot be removed from the network*; they can only move from a node to a *neighbouring node*. This assumption is obviously motivated by the fact that we are considering software agents that are able to move only on the edges of the network. More precisely, we consider the *contiguous, monotone, node search problem* first studied in [1] where (1) the removal of agents is not allowed, (2) at any time of the search strategy, the set of clean nodes forms a connected subnetwork, and (3) a clean node cannot be recontaminated.

The contiguous assumption considerably changes the nature of the problem and the classical results on node and edge search do not generally apply. The problem of finding the optimal number of agents is still NP-complete for arbitrary graphs, and has been shown to be solvable with a linear number of moves and with the minimal number of agents only in trees, and meshes [1, 6].

An investigation on the relationship between the contiguous model and the classical one for graph search has been studied in [2]. In particular, the authors have shown that the *search number* $s(G)$ (*contiguous monotone search number* $cms(G)$) of a graph G, i.e., the smallest number of agents for which a (contiguous monotone) searching strategy exists is related to $cms(G)$ as follows: $s(G) \leq cms(G)$.

1.3 Our Results

In this paper we study the contiguous monotone node search problem in one of the most common topology for interconnection network: the hypercube.

The basic assumption we make is that the agents know the topology, and can communicate through whiteboards located at the nodes. We then consider increasingly more powerful models adding assumptions to the power of the agents.

In the first model the only knowledge that an agent has is the information written on the local whiteboard at its current location, and the strategy relies on a coordinator that organizes the moves of the other agents. In the second model an agent can "see" also the state of its neighbours; i.e.,

	Agents	Time	Moves
Coordinator	$O(\frac{n}{\sqrt{\log n}})$	$O(n \log n)$	$O(n \log n)$
Visibility	$n/2$	$\log n$	$O(n \log n)$
Visibility and Cloning	$n/2$	$\log n$	$n-1$
Synchronous with Cloning	$n/2$	$\log n$	$n-1$

Table 1: Search in a d-dimensional Hypercube H_d, with $n = 2^d$ nodes.

it can see whether a neighbouring node is guarded by an agent, whether it is clean, or contaminated. Notice that, in some mobile agent systems, the visibility power could be easily achieved by "probing" the state of neighbouring nodes before making a decision. The strategy for this model assumes that the agent takes advantage of its visibility power and can move autonomously. In the third model an agent can both "see" the state of its neighbours and thus move autonomously, and has also "cloning" capabilities, i.e., can create copies of itself. Finally, in the last model an agent has both cloning capabilities and can autonomously move in a synchronous setting, i.e., every action it performs takes one unit of time. Our results are summarized in Table 1.

The bound on the number of agents that we obtain in our first strategy ($O(\frac{n}{\sqrt{\log n}})$) is optimal. An interesting consequence of this result is the derivation of the search number for the hypercube in the classical model, which was unknown. In fact, in the classical model the lower bound is the same, thus our strategy is optimal also when the agents are allowed to "jump". The hypercube is then another topology where "jumping" does not help reducing the number of agents, another example is the tree [1].

In all the other strategies time is optimal, but the number of agents is not. Finally, when the agents have both cloning and visibility, and when they have cloning in a synchronous environment, both time and number of moves are optimal.

2 Definitions and Terminology

A d-dimensional hypercube H_d has $n = 2^d$ nodes and $d2^{d-1}$ edges. Each node corresponds to an d-bit binary string, and two nodes are linked with an edge if and only if their binary strings differ in precisely one bit.

Let $G = H_d = (V, E)$ be a hypercube; let $n = |V|$ be the size of G, $E(x)$ be the edges incident on $x \in V$. If $(x, y) \in E$ then x and y are said to be neighbors. At each node x, there is a distinct label associated to each of its incident edges; the label between node x and node z is the position of the bit in which the corresponding binary strings differ. Let $\lambda_x(x, z)$ denote the label associated at x to edge $(x, z) \in E(x)$.

Operating in (G, λ) is a team of autonomous mobile agents. The agents can move from a node to a neighboring node in G, have computing capabilities and computational storage ($O(\log n)$ bits suffice for all our algorithms), obey the same set of behavioral rules. In the *asynchronous* environment every action the agents perform (i.e., computing, moving), takes a finite but otherwise unpredictable amount of time; in the *synchronous* environment we assume that local computations are instantaneous, and it takes one unit of time (one step) to move from node to neighbouring node. The agents know that the topology they are searching is a hypercube.

Each node has a local storage area called *whiteboard* ($O(\log n)$ bits of memory suffice for all our algorithms). It is through the whiteboards that agents communicate, in fact they can read from and write on the whiteboards. Access to a whiteboard is gained fairly in mutual exclusion. In particular, the initial information contained in the whiteboard of a node are: its Id (binary string), and the label of the incident ports.

At each point in time, a node x of the hypercube could be in one of the following three states: *guarded*, if an agent is currently on x; *clean*, if an agent passed by x and all its neighbours are either clean or guarded; *contaminated* otherwise.

Initially, all agents are in the same node, (the *homebase*), which is then guarded, and all the other nodes are contaminated; a node cleaning strategy consists of a sequence of movements of the agents along the edges of the network.

We say that a cleaning strategy is *monotone* if once a node is clean, it is never going to be contaminated again. In the following we will be interested only in monotone strategies.

Let us organize the hypercube H_d in $d+1$ levels and let level $i = 0, 1, \ldots, d$ consist of all the nodes whose binary representation contains i ones. Clearly, all the nodes at level i are connected only to nodes of level $i-1$ and to those of level $i+1$.

We consider a special breadth first spanning tree of the hypercube rooted at the source (node (00..00)). Let $m(x)$ denote the position of the most significant bit of x, then there is an edge in the spanning tree between x and all the nodes in the next level whose bit in which they differ is in a position higher than $m(x)$ (see Figure 1). This spanning tree is also called *broadcast tree* because it is employed to perform optimal broadcast in the hypercube: a node x receiving a message from dimension i will forward it to all nodes connected by dimension $j > i$.

The resulting spanning tree (also known as heap queue) has the following structure:

Definition 1. *A heap queue $T(d)$ is a rooted tree recursively defined as follows:*
- $T(0)$ is a leaf;
- $T(1)$ is a node with one child;
- $T(k)$ is a node with k children of type $T(0), \ldots T(k-1)$.

It is very well known that a broadcast spanning tree of a hypercube of size n is a heap-queue $T(\log n)$. In fact, level 0 contains the source $(00\ldots0)$ of the tree (which is of type $T(d)$ where $d = \log n$ is the degree of the hypercube).

To easily refer to the neighborhood of a node, the neighbors are grouped into either smaller neighbors or bigger neighbors as defined below. Let x, y be two neighbouring nodes; i.e., $(x, y) \in E(x)$.

Definition 2. *Node y is called a smaller neighbor of node x if $\lambda_x(x, y) \leq m(x)$; y is called a bigger neighbor of x if $\lambda_x(x, y) > m(x)$.*

Notice that the bigger neighbors of x are the children of x in the broadcast tree.

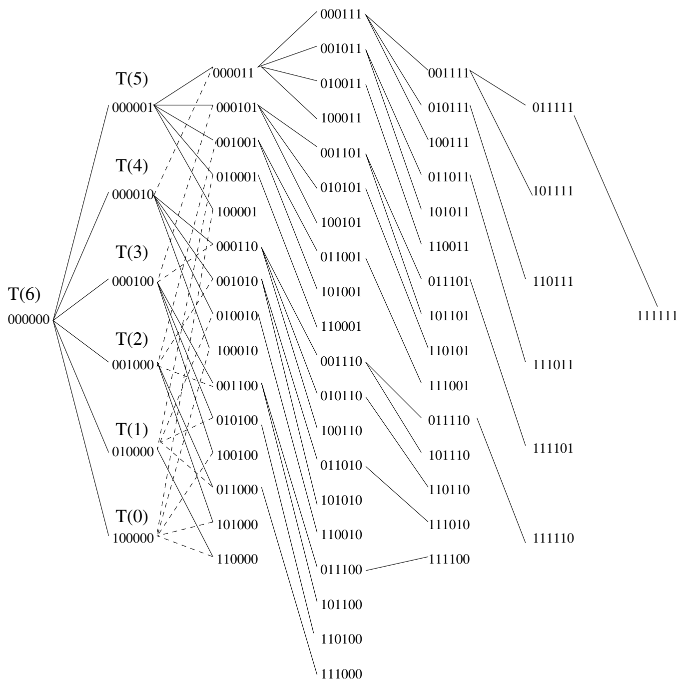

Figure 1: The broadcast tree $T(6)$ of the hypercube H_6. Normal lines represent edges in $T(6)$, dotted lines (only partially shown) the remaining edges of H_6.

3 The Strategies

In this section we propose four different strategies for the capture of the intruder in the hypercube. As far as we know, this topology has never been investigated neither in the classical model, nor in the contiguous one.

First of all, let us state a lower bound, that holds for all models, on the number of agents needed to clean the hypercube:

Theorem 1. *To solve the contiguous monotone search problem $\Omega(\frac{n}{\sqrt{\log n}})$ agents are required.*

Proof. It is known that the node search number of a graph (in the classical model) is equal to its pathwidth plus one [8]. From [3, 4] we know that the pathwidth of a hypercube is $\Theta(\frac{n}{\sqrt{\log n}})$.

Since a lower bound in the classical model is obviously a lower bound also in our mode, the theorem follows. $\square$

3.1 Agent-Optimal Strategy

For the ease of discussion, in this section we assume that the degree of the hypercube d is even with $d \geq 4$, the same bounds with minor technical modifications are obtained for odd degrees.

The first strategy we present is optimal in terms of number of agents, in fact it uses $O(\frac{n}{\sqrt{\log n}})$ agents.

The main idea of our strategy is that all the agents start the procedure at the same homebase. One agent acts as a leader and coordinates the activities of the other agents. Exploiting the knowledge of the topology, the coordinator makes optimal use of the agents making them visit all nodes and while protecting the system from the intruder recontamination.

3.1.1 Basic Properties.

Before providing a detailed description of our strategy we first introduce some basic properties that we will need in the following:

Property 1. *Let $T(d)$ be a broadcast tree for the hypercube. The number of nodes at level l is $\binom{d}{l}$.*

PROOF By construction the nodes at level l are those containing l ones in their binary representation. Since the representation contains d bits, there are exactly $\binom{d}{l}$ such nodes. ∎

Property 2. *At level 0 there is a unique node of type $T(d)$. At level $l > 0$ there are $\binom{d-k-1}{l-1}$ nodes of type $T(k)$ with $0 \le k \le d - l$.*

PROOF Nodes of type $T(k)$ start with k 0s followed by a one. Since they are at level l they must contain $l - 1$ 1s in the last $d - k - 1$ positions. ∎

Property 3. *Let $T(d)$ be a broadcast tree for the hypercube. Level 0 has no leaves. The number of leaves at level $l > 0$ is $\binom{d-1}{l-1}$.*

PROOF By definition of broadcast tree, the leaves are those nodes whose d-bit representation start with 1 (i.e., the nodes that do not have bigger neighbours at the next level). Thus, they contain $l - 1$ 1s in the remaining $d - 1$ bits. ∎

Property 4. *Let $T(d)$ be a broadcast tree for the hypercube. The total number of leaves is 2^{d-1}.*

PROOF By definition of broadcast tree, the leaves are those nodes whose d-bit representation start with 1. Thus, they are half of the total number of nodes. ∎

In the following we will sometimes use an alternative labeling of the nodes of the hypercube. Given a node with binary representation x, we label it also with a vector $\widehat{x}$ defined as follows: $\widehat{x} =< i_1, i_2, ..., i_l >$ where i_k are the positions of the 1 bits in x for $1 \le k \le l$, and $i_k > i_{k-1}$. For example, node (001110) is also labeled as $< 2, 3, 4 >$. In the following we will consider the nodes at each level of the broadcast tree lexicographically ordered according to this labeling, (this is the order in which nodes appears in each level in Figure 1), and not according to their binary string.

3.1.2 Cleaning strategy.

The agents are all identical; one of them, however, will act as *coordinator* for the entire cleaning process. The coordinator can be locally selected by the agents by accessing the whiteboard; the first that gains access will become the coordinator.

The cleaning strategy is carried out on the broadcast tree. The main idea is to place enough agents in node $(00 \ldots 00)$ and then have them move level by level. The agents will move on the broadcast tree leaded by the coordinator in such a way that, during their movements, the intruder cannot enter the area already cleaned. While cleaning, the agents (except the coordinator) will move only along the edges of the broadcast tree.

At the beginning, all the agents are available at the root. We call the agents available at the root, *set of available agents*.

Algorithm CLEAN.

1. From the root to level 1

1.1 - The coordinator guides for d times a distinct agent from the root of the tree to each of its d children $T(d-1)\dots T(0)$, each time returning back to the root.

2. From level $l \geq 1$ to level $l+1 \leq d$. [level l has one agent per node]

2.1 - Before starting to clean nodes at level $l+1$, the coordinator moves back to the root to collect the agents needed for completing the cleaning of level $l+1$ (i.e., $k-1$ agents per node of type $T(k)$ except for the nodes of type $T(1)$ and $T(0)$ which do not require any extra agent). The root sends $k-1$ additional agents, in no specific order, to each node of type $T(k)$, $k>1$ at level l.

2.2 - When k agents are on a node of type $T(k)$ at level l, they are sent down in the broadcast tree to the children at level $l+1$ guided by the coordinator. Let $\widehat{n_1},\dots,\widehat{n_m}$ ($m=\binom{d}{l}$) be the nodes of level l lexicographically ordered. The coordinator sequentially chooses each node at level l following this lexicographical order and node by node guides on each outgoing edge of the broadcast tree one agent to level $l+1$.

2.3 - When a leaf of level l is reached by the coordinator, the agent on it becomes available and goes back to the root. Notice that when the coordinator reaches the last node of level l, the only active agents are the ones covering level $l+1$.

Notice that the whiteboard is used for any communication between the coordinator and the agents. In particular, during the strategy the whiteboard stores the labels of the incident ports and the current number of agents present at the node; it is accessed by the coordinator to decide when and where to send the agents to the next level. No more that $O(\log n)$ bits are required for this local storage.

Figure 2 shows the order in which the nodes are cleaned by the agents leaded by the coordinator.

3.1.3 Correctness and Analysis.

Correctness. We now prove that Algorithm CLEAN is correct; i.e., that all nodes will be cleaned and that once a node has been cleaned, it will never be recontaminated.

Let x be a node of level l, $N(x)$ denote the neighbours of x at level $l+1$ and $NT(x)$ denote the children of x at level $l+1$ in the broadcast tree. Notice that $N(x)$ includes $NT(x)$ and possibly some other neighbours at level $l+1$.

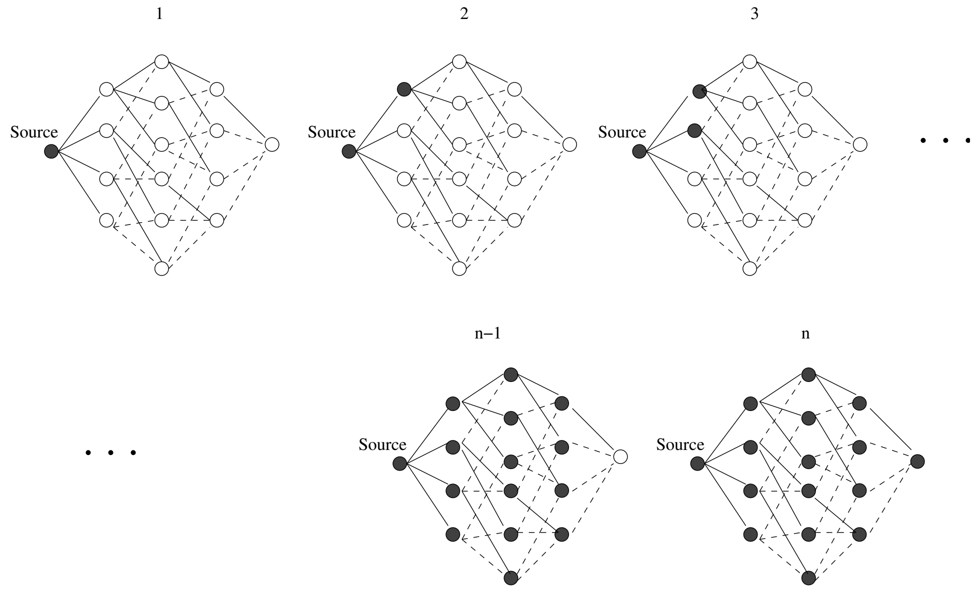

Figure 2: Algorithm CLEAN on a 4-dimensional hypercube. The nodes get cleaned sequentially.

Lemma 1. *If $z \in N(y) - NT(y)$ then $z \in NT(x)$ for some x such that $\widehat{x} < \widehat{y}$.*

PROOF Let h be the position of the most significant bit of y. Since $z \notin NT(y)$ we know that it does not contain a 1 bit in position higher than h. However, $z \in N(y) - NT(y)$, i.e., it is a neighbour of y (not in the broadcast tree) at level $l + 1$, therefore its binary representation contains an extra bit in a position lower than h. Without loss of generality, let us assume that $k < h$ is such a position (so the the k-th bit of y is 0 while the k-th bit of z is 1). Consider now a node x whose binary representation is the same as the one of y except for the h-th bit, which is 0 and the k-th bit, which is instead 1. Node x has the same number of 1s as node y. So node x and y are at the same level. Moreover, since the k-th bit in x is 1 but the k-th bit in y is 0, by the definition of lexicographical order, we have $\widehat{x} < \widehat{y}$. We have that $z \in NT(x)$ because they differ in just one bit (the h-th) and moreover the most significant bit of z (the h-th) is higher than the most significant bit of x. ∎

Lemma 2. *In Algorithm* CLEAN*, when agents leave a node x at level l (leaving it unguarded) all neighbours of x are either clean or guarded.*

PROOF This is clearly true for the one node of level 0. Assume it is true for all nodes of level $0 \leq j < i$ and consider level i.
Nodes at level i are only connected to nodes at level $i-1$ and $i+1$. When nodes at level i are sending agents to level $i + 1$, by induction hypothesis and the cleaning strategy, all nodes at level $i - 1$ are clean. Let us now assume that cleaning occurs at node y of level i. For any $z \in N(y) - NT(y)$, by lemma 1, $\exists x$ such that $z \in NT(x)$ and $\widehat{x} < \widehat{y}$. According to the cleaning strategy, the coordinator approaches the nodes in lexicographical order at each level. So before the coordinator reaches node y agents on node x are already sent to level $i + 1$ and all $z \in N(y) - NT(y)$ are already guarded by an agent each. If y is a node of type $T(j)$ where $j \leq 1$, by step 2.1 of Algorithm CLEAN, from

the set of available agents on the homebase, enough agents are sent to node y to clean the children of y in the broadcast tree. If y is a node of type $T(1)$, the agent on it is enough to clean the only child at level $i+1$. After all agents on y are sent down to the children of y at level $i+1$, each of $NT(y)$ is guarded by an agent and all neighbors of y are either clean or guarded by an agent. If y is a leaf, $NT(y)$ is empty. When the coordinator reaches it, all neighbours of y at level $i+1$ are guarded and all neighbours at level $i-1$ are clean. ∎

Theorem 2. *The cleaning process of Algorithm* CLEAN *decontaminates all nodes. During the execution clean nodes can not be recontaminated.*

PROOF First of all, by the cleaning strategy, the cleaning is performed level by level. So when it reaches level d, all nodes have been cleaned. The fact that a clean node will not be recontaminated directly follows from lemma 2 ∎

Complexity. We remind that we are considering here hypercubes with even dimension $d \geq 4$. We first calculate the number of agents needed to perform the cleaning of the hypercube with Algorithm CLEAN.

Theorem 3. *In a d-dimensional hypercube, Algorithm* CLEAN *employs* $\Theta(\frac{n}{\sqrt{\log n}})$ *agents to clean the network, which is optimal.*

We will prove the theorem by calculating the number of agents needed to clean from level l to level $l+1$, and by showing that the maximum number of agents is needed when cleaning from level $\frac{d}{2}-1$ to $\frac{d}{2}$ and from level $\frac{d}{2}$ to level $\frac{d}{2}+1$, where the corresponding number will be shown to be $\binom{d}{\frac{d}{2}} + \binom{d-1}{\frac{d}{2}-2} + 1 = O(\frac{n}{\sqrt{\log n}})$.
We first calculate the number of agents that are taken from the set of available agents before performing the cleaning from level l to level $l+1$.

Lemma 3. *In Algorithm* CLEAN*, before cleaning from level* $l \geq 1$ *to level* $l+1 \leq d$*,* $\binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}$ *extra agents are fetched from the root by the coordinator.*

PROOF In step 2.1 of Algorithm CLEAN, $k-1$ extra agents for a node of type $T(k)$ are sent from the root. By property 2, at level $l \geq 1$ there are $\binom{d-k-1}{l-1}$ nodes of type $T(k)$ with $0 \leq k \leq d-l$. So totally $\sum_{k=2}^{d-l}(k-1)\binom{d-k-1}{l-1}$ extra agents are sent from the root to level l while cleaning from level l to level $l+1$. Note that first choosing $i = k-1$ and then $L = l-1$ we have:
$\sum_{k=2}^{d-l}(k-1)\binom{d-k-1}{l-1} = \sum_{i=1}^{d-l-1} i\binom{d-(i+1)-1}{l-1} = \sum_{i=1}^{d-L-2}\binom{i}{1}\binom{d-i-2}{L}$.
Observe now that given $a, b \in \mathbb{N}$ we have $\binom{a}{b} = 0$ for $a < b$. Therefore:
$\sum_{i=1}^{d-L-2}\binom{i}{1}\binom{d-i-2}{L} = \sum_{i=1}^{d-2}\binom{i}{1}\binom{d-i-2}{L} = \sum_{i=0}^{d-2}\binom{i}{1}\binom{d-i-2}{L}$.
Referring to ([7]) we have that: $\sum_{i=0}^{d-2}\binom{i}{1}\binom{d-i-2}{L} = \binom{d-1}{L+2}$; thus,
$\sum_{i=0}^{d-2}\binom{i}{1}\binom{d-i-2}{L} = \binom{d-1}{l+1} = \binom{d}{l+1} - \binom{d-1}{l} = \binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}$. ∎

We now show that:

Lemma 4. $\binom{d}{\frac{d}{2}}+\binom{d-1}{\frac{d}{2}-2}+1$ *agents are enough in Algorithm* CLEAN *to perform the cleaning from level* $l \geq 1$ *to level* $l+1 \leq d$.

PROOF By induction.
There are enough agents to clean level 1 because only d agents are needed and $d < \binom{d-1}{\frac{d}{2}-2}+\binom{d}{\frac{d}{2}}+1$ for $d \geq 4$.
Let us assume that there are enough agents to clean level l, $l \geq 1$, using our strategy. At this point we have $\binom{d}{l}+1$ active agents (where the coordinator is counted too), every node of level l is guarded by one agent; all the other agents are available. We now show that there are enough agents to clean level $l+1$ using our strategy.
By lemma 3, before cleaning from level l to level $l+1$, the coordinator collects extra $\binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}$ agents. This number has been calculated so to give exactly k agents for each tree of type $T(k)$, which is just enough to proceed with the cleaning of level $l+1$. In fact, by induction hypothesis, each node of level l is already guarded by one agent before the extra agents come. So in total, $\binom{d}{l}$ + 1 + $\binom{d}{l+1}-\binom{d}{l}+\binom{d-1}{l-1}$ = $\binom{d}{l+1}+\binom{d-1}{l-1}+1$ agents are active at level l. By our strategy, the $\binom{d-1}{l-1}$ agents on the leaves do not participate in the cleaning of level $l+1$, but the other $\binom{d}{l+1}$ are just enough to move to level $l+1$ on the broadcast tree.
In addition, it is well known that:

$\max_{1\leq l\leq d-1} \binom{d}{l+1}+\binom{d-1}{l-1} = \binom{d}{\frac{d}{2}+1}+\binom{d-1}{\frac{d}{2}-1} = \binom{d}{\frac{d}{2}}+\binom{d-1}{\frac{d}{2}-2}$ for $l=\frac{d}{2}$ *or* $l=\frac{d}{2}-1$, respectively.

Therefore, $\binom{d}{\frac{d}{2}}+\binom{d-1}{\frac{d}{2}-2}+1$ is the maximum number of agents required by the procedure and corresponds to the cleaning of the central level (this number corresponds to the nodes of level $\frac{d}{2}$, plus the leaves of the broadcast tree at level $\frac{d}{2}-1$). Finally observe that, from the Stirling's approximation $\binom{d}{\frac{d}{2}} = O(\frac{4^d}{\sqrt{d}})$ which is $O(\frac{n}{\sqrt{\log n}})$; thus $\binom{d}{\frac{d}{2}}+\binom{d-1}{\frac{d}{2}-2}+1 = O(\frac{n}{\sqrt{\log n}})$. ∎

We now calculate the total number of moves needed for the entire process.

Theorem 4. *The total number of moves performed by the agents in Algorithm* CLEAN *is* $O(n \log n)$.

PROOF To compute the global number of moves we have to take into account the ones performed by the agents and those performed by the coordinator.
The number of moves performed by the agents :
It takes $2l$ moves for an agent to arrive at a leaf of level l from the root and go back to the root. By property 3, there are $\binom{d-1}{l-1}$ leaves at level l. So totally, there are $\sum_{l=1}^{d} 2l\binom{d-1}{l-1}$ moves performed by the agents. To compute this quantity first note that $\binom{d}{l}$ is the number of nodes at level l, thus summing up over all levels we obtain, $\sum_{l=0}^{d}\binom{d}{l} = 2^d = n$. It follows that $\sum_{l=1}^{d}\binom{d-1}{l-1} = \sum_{l=0}^{d-1}\binom{d-1}{l} = 2^{d-1} = \frac{n}{2}$.
We have now to compute:

$$\sum_{l=1}^{d} l\binom{d-1}{l-1} = 1\binom{d-1}{0}+...+(\tfrac{d}{2}-1)\binom{d-1}{\frac{d}{2}-2}+\tfrac{d}{2}\binom{d-1}{\frac{d}{2}-1}+(\tfrac{d}{2}+1)\binom{d-1}{\frac{d}{2}}+(\tfrac{d}{2}+2)\binom{d-1}{\frac{d}{2}+1}+...+d\binom{d-1}{d-1}.$$

Using the property that $\binom{a}{b} = \binom{a}{a-b}$ we may group terms in pair and obtain:

$$\sum_{l=1}^{d} l\binom{d-1}{l-1} = (d+1)\binom{d-1}{0}+...+(d+1)\binom{d-1}{\frac{d}{2}-1} = (d+1)\sum_{l=0}^{\frac{d}{2}-1}\binom{d-1}{l}.$$

We already know that: $\sum_{l=0}^{d-1}\binom{d-1}{l} = 2^{d-1} = \frac{n}{2}$; we also know that: $\sum_{l=0}^{d-1}\binom{d-1}{l} = 2\sum_{l=0}^{\frac{d}{2}-1}\binom{d-1}{l}$, so, $(d+1)\sum_{l=0}^{\frac{d}{2}-1}\binom{d-1}{l} = (d+1)2^{d-2} = \frac{n}{4}(\log n + 1)$.
Finally, the total number of moves performed by the agents is:

$$\sum_{l=1}^{d} 2l\binom{d-1}{l-1} = \frac{n}{2}(\log n + 1) = O(n \log n).$$

The number of moves performed by the coordinator:

1. Go to the root to get more agents. Totally, there are $\sum_{l=1}^{d-2} l = \frac{(d-2)(d-1)}{2} = O(\log^2 n)$ moves.
2. Go to the first node of each level. Totally, there are $\sum_{l=1}^{d-2} l = \frac{(d-2)(d-1)}{2} = O(\log^2 n)$ moves.
3. Navigate within each level to get to the next node. Recalling that the procedure is run on an hypercube structure we obtain that at level l, the coordinator needs to navigate at most $2l$ edges if $l \leq \frac{d}{2}$, otherwise $2d - 2l$ edges to reach the next node at the same level. So totally the number of moves is at most $\sum_{l=1}^{\frac{d}{2}} 2l\binom{d}{l} + \sum_{l=\frac{d}{2}+1}^{d-1}(2d-2l)\binom{d}{l} = 4\sum_{l=1}^{\frac{d}{2}-1} l\binom{d}{l} + d\binom{d}{\frac{d}{2}} = O(n \log n)$.
4. Go down with each agent to clean a node at the next level in the broadcast tree and then come back. Each edge of the broadcast tree is traveled twice by the coordinator. So totally there are $2(2^d - 1) = 2(n-1)$ moves.

Totally, the number of moves performed by the cleaning process is $O(n \log n)$. ∎

We now consider the ideal time complexity of the cleaning strategy. We remind that we are in this case we are working in an asynchronous environment, so we consider the ideal time complexity (i.e., by assuming that it takes one unit of time for an agent to traverse an edge).

Theorem 5. *The cleaning strategy of Algorithm* CLEAN *takes $O(n \log n)$ time units.*

PROOF The cleaning process is carried out sequentially by the coordinator. The time required is then equal to the number of moves of the coordinator. ∎

Note. Notice that, for the ease of calculations and description of algorithm CLEAN, we have assumed that the hypercube has an even number of dimensions. Let us briefly discuss what happens if the number of dimensions is odd. Since the algorithm does not depend on the fact that d is odd or even, the correctness still holds. What is affected is only the maximum number of agents employed.

In fact, in the proof of Lemma 4 we have that the number of agents employed is:
$\max_{1 \leq l \leq d-1}\binom{d}{l+1} + \binom{d-1}{l-1} + 1$. When d is even we have seen that this number corresponds to the number of nodes $\binom{d}{\frac{d}{2}}$ in the maximum level, plus the number of leaves $\binom{d-1}{\frac{d}{2}-2}$ in the previous level, plus 1. When d is odd, there are two central levels ($\frac{d+1}{2}$ and $\frac{d-1}{2}$) of the broadcast tree, with the same number of nodes. In this case, $\max_{1 \leq l \leq d-1}\binom{d}{l+1} + \binom{d-1}{l-1} + 1$ corresponds to the number of nodes $\binom{d}{\frac{d+1}{2}}$ in level $\frac{d+1}{2}$ plus the number of leaves $\binom{d-1}{\frac{d+1}{2}-1}$ (equivalently $\binom{d-1}{\frac{d-1}{2}}$) in the previous level, plus 1. This value is clearly still $O(\frac{n}{\sqrt{\log n}})$.

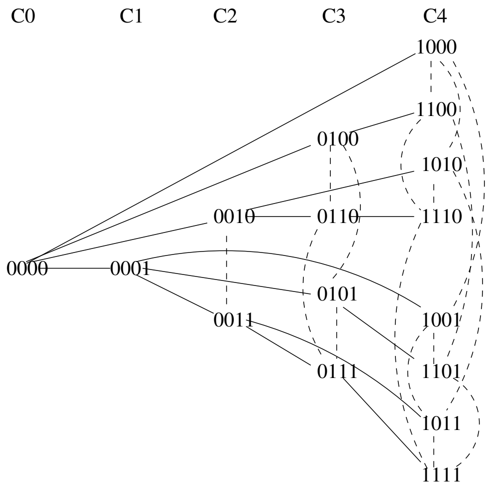

Figure 3: The 4-dimensional hypercube H_4 in the new group structure.

3.2 Visibility Power: Time-Optimal Strategy

In this section we consider the node search problem in the hypercube in a new different model. We assume that an agent located at a node can "see" whether its neighboring nodes are clean or guarded or contaminated. This capability could be easily achieved if the agents have also communication power and send a message (e.g., a single bit) to their neighbouring nodes after cleaning a node or guarding a node. The interesting aspect of this model is that this extra capability enables agents to correctly act without the need of being coordinated by the coordinator.

In the following we will then make the following assumption on the agent *visibility*:

An agent located at node x can see the state of the neighbours $N(x)$.

Thus, as we will see later in this section, the visibility assumption allows the agents to make their own decision regarding the action to take solely on the basis of their local knowledge.

3.2.1 Basic Properties

We now introduce some basic properties that we will need in the following. First, we group the nodes of the hypercube based on the position of the most significant bit in their binary string.

Formally, let C_i be the set of nodes whose most significant bit is in the i-th position (see Figure 3).

Property 5. *Exactly one node is in C_0; 2^{i-1} nodes are in C_i for $0 < i \leq d$;*

PROOF By definition, only node $(00..00)$ is in C_0. The nodes in C_i have the most significant bit in the i-th position. There are 2^{i-1} nodes with most significant bit in the i-th position. Hence C_i has 2^{i-1} nodes. ∎

Property 6. *All the leaves of the broadcast tree are in C_d.*

Let node x be any node in C_i. Since C_0 has only one node (00..00), which has no smaller neighbors, for the following property we only consider x in C_i where $i > 0$.

Property 7. *Consider a node x in C_i, $i > 0$. One smaller neighbor of node x is in C_j (where $j < i$), the other smaller neighbors, if any, are in C_i and all its bigger neighbors, if any, are in C_k (where $k > i$).*

PROOF By definition, the most significant bit of node x is in the i-th position. The labels of its neighbors differ only in one bit. If a neighbor y is different from x in the l-th position for $l < i$, then y is a smaller neighbor of x and y has a 1 bit in the i-th position. By definition, y is in C_i too. If the neighbor y is different from x in the i-th position, then the most significant bit of y must be in position j, $j < i$. So y is in C_j. By definition, y is a smaller neighbor of x too. Since there is only one neighbor different from x in the i-th position, we have that one smaller neighbor of node x is in C_j, $0 \leq j < i$ and the other smaller neighbors are in C_i. If the neighbor y is different from x in the k-th position where $k > i$, then the k-th position of y must be 1 because the most significant bit of x is in the i-th position. By definition, y is the bigger neighbor of x and it is in C_k where $k > i$. ∎

Let node x be any node in C_i. Since node (00..00) in C_0 has no smaller neighbor and node (00..001) in C_1 has one smaller neighbor in C_0, but no smaller neighbor in C_1, we only consider node x in C_i where $i > 1$ for the following property.

Property 8. *There must exist at least one smaller neighbor y of node x such that y is in C_i and a smaller neighbor z of y is in C_{i-1}.*

PROOF By property 7, all smaller neighbors (except one) of node x are in C_i, i.e., have bit 1 in the i-th position and differ from x only in the j-th position, where $j < i$. Consider a smaller neighbour y that differs from x only in position $i - 1$, if x has a 0 in this position, or otherwise in a position $j < i - 1$. In both cases y has a 1 both in position i and $i - 1$. Consider now a smaller neighbor z of y that differs from y only in the i-th position. z has a 0 in position i and a 1 in position $i - 1$. Thus, the most significant bit of z is in position $i - 1$ and so z belongs to C_{i-1}. ∎

3.2.2 Cleaning strategy

All the agents are initially located at the source; they are identical and autonomous, and they all follow the same local rule.

The agents are still moving on the broadcast tree, but they do not have to follow the order imposed by the coordinator like in the previous section. In fact, the agents on node x can proceed to clean the children of x in the broadcast tree when they "see" that the other neighbours of x are either clean or guarded. More precisely, the agents on node x can proceed to clean the bigger neighbours of x when they "see" that all the smaller neighbours of x are either clean or guarded.

Algorithm CLEAN WITH VISIBILITY.

Rule for the agents on node x of type $T(k)$ $(0 \leq k \leq d)$:

1. Wait until 2^{k-1} agents are on x.
2. Wait until all the smaller neighbours of x are clean or guarded.
3. One agent moves to the bigger neighbour of type $T(0)$; 2^{i-1} agents move to each of the bigger neighbours of type $T(i)$, for $0 < i < k$; if there are no bigger neighbours, terminate.

Notice that the whiteboard is used for storing the status of the node (*contaminated*, *clean*, *guarded*), and the current number of agents present at the node. During the strategy the agents can access the local whiteboard and the whiteboards of the neighbours. The whiteboard accessing determines to which node each agent has to move. Also in this case, no more that $O(\log n)$ bits are required for the whiteboard.

Figure 4 shows the order in which the nodes get cleaned with our strategy. As opposed to the strategy of the previous section, nodes are not cleaned sequentially; several nodes, in fact, could be cleaned independently.

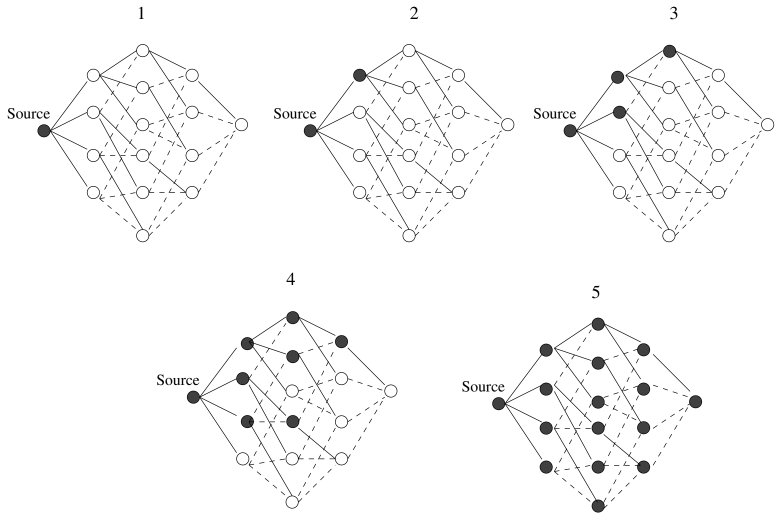

Figure 4: The execution of Algorithm CLEAN WITH VISIBILITY on a 4-dimensional hypercube. Nodes get cleaned independently.

3.2.3 Correctness and Complexity

Correctness.

Theorem 6. *The total number of agents needed to clean the d-dimensional hypercube using Algorithm* CLEAN WITH VISIBILITY *is $\frac{n}{2}$.*

PROOF By definition Algorithm CLEAN WITH VISIBILITY is sending from level 0 to level 1 one agent for $T(0)$ and 2^{i-1} agents for each $T(i)$ for a total of $1 + \sum_{i=1}^{d-1} 2^{i-1} = 1 + \sum_{i=0}^{d-2} 2^i =$

$2^{d-1} = \frac{n}{2}$ agents. Moreover, a node of type $T(k)$ receives 2^{k-1} agents and 2^{k-1} is exactly the number of agents needed to continue the cleaning strategy, in fact, $2^{k-1} = 1 + \sum_{i=1}^{k-1} 2^{i-1}$. Thus, with $\frac{n}{2}$ agents the strategy can be completed. ∎

We now prove that Algorithm CLEAN WITH VISIBILITY is correct; i.e., that the network is clean and once a node has been cleaned, it will never be recontaminated.

Lemma 5. *In Algorithm* CLEAN WITH VISIBILITY, *when agents leave a node in C_i (leaving it unguarded) all its smaller neighbours are either clean or guarded.*

PROOF Let us consider a node x in C_i. By the cleaning strategy, when an agent arrives at node x, it cleans the node. By theorem 6 every node will have eventually enough agents to continue the cleaning process, and by definition of the cleaning rule, the agents on x move to the bigger neighbours only when all other neighbours are clean or guarded. ∎

Theorem 7. *During the cleaning process of Algorithm* CLEAN WITH VISIBILITY *the agents clean all nodes and a clean node will not be recontaminated.*

PROOF First of all, by the cleaning strategy, the edges and nodes traversed by the agents form the broadcast tree. All the nodes are visited by an agent. The fact that a clean node will not be recontaminated directly follows from lemma 5. ∎

Complexity. We now consider the time complexity of the cleaning strategy.

Theorem 8. *Cleaning the entire network with Algorithm* CLEAN WITH VISIBILITY *takes $O(\log n)$ time units, which is optimal.*

PROOF We will prove it by showing that at time i, all nodes in C_i are clean; only the agents in C_i can move to clean the bigger neighbors, which are in C_j for $j > i$. We prove it by induction.
Base case: at time $i = 0$, all the agents are placed on node $(00...00)$ (the source). First notice that, since there is no agent on any other node at time 0, only the agents in C_0 might move at this time. By the cleaning strategy, the agents clean the source, then they move to clean the d bigger neighbors, which are in C_j for $0 < j \leq d$. Node $(00..00)$ becomes clean at time 0; obviously, no recontamination can occur to it. The claim then holds for $i = 0$.
Assume the claim is true up to time i, $i \geq 0$. We show that it holds at time $i + 1$.
By induction hypothesis and theorem 7 all the nodes in C_k are clean for any $0 \leq k \leq i$. The agents ever on them have left. Let node x be an arbitrary node in C_{i+1}. By property 7, exactly one smaller neighbour of x is in C_k for $k \leq i$. By induction hypothesis, at time k, the agents arrive at x and clean it upon arrival. So at time $i+1$, every node in C_{i+1} is guarded by at least an agent; and, thus, all smaller neighbours of any node in C_{i+1} are clean or guarded. So at time $i + 1$ every agent in C_{i+1} executes the algorithm; it cleans the bigger neighbors which, by property 7, are in C_j with $j > i + 1$. From the fact that the nodes in C_{i+1} are already cleaned by their guarding agents upon arrival, and that by theorem 7 a clean node will not be recontaminated, we know that at time $i + 1$ the nodes in C_{i+1} become clean after the agents on them move.
Notice that, by our cleaning algorithm, the other agents in C_j for $i + 1 < j \leq d$ cannot move because one or more of their smaller neighbours are not guarded. Let us consider any node y in C_j

on which there are agents. By property 8, there exists one smaller neighbor z of y which is in C_j too and a smaller neighbor w of z which is in C_{j-1}, $j-1 \geq i+1$. We know that the agents on z come from its smaller neighbor w which is in C_{j-1}. If $j-1 = i+1$, in other words, w is in C_{i+1}, the agents on w move at time $i+1$. So at time $i+1$ no agent is on z yet. If $j > i+1$, even if there are agents on w, by induction hypothesis, they have not moved before time $i+1$. Hence, in any case, z is not guarded at time $i+1$ and the agents on y cannot move because at least one of its smaller neighbors is not guarded.

The time complexity is clearly optimal, since $\log n$ is the diameter of the hypercube and the agents are initially all located in the same node. ∎

We now calculate the total number of moves performed by the agents.

Theorem 9. *The number of moves performed by the agents in Algorithm* CLEAN WITH VISIBILITY *for the entire cleaning is $O(n \log n)$.*

PROOF All the agents start from the source and each terminates on a leaf. There are $\binom{d-1}{l-1}$ leaves at level $l > 0$, thus the total number of moves is: $\sum_{l=1}^{d} l \binom{d-1}{l-1} = O(n \log n)$ (the calculation is similar to the one of the proof of theorem 4). ∎

3.3 Observations About Other Models

Visibility and Cloning. In this section we consider the node search problem in the hypercube in a model in which agents have both visibility power and *cloning* capabilities, the last being the agent capacity of creating a copy of itself. No coordinator exists to centrally control the cleaning. All the agents are autonomous and they all follow the same algorithm.

Initially, one agent is placed at node $(00...00)$. It cleans node $(00...00)$ and clones $d-1$ new agents. In this way d agents are available on the node and then one agent per edge is sent to clean each of the d neighbors. When an agent arrives at a node, it cleans this node and then executes the following algorithm.

Algorithm CLEAN WITH VISIBILITY AND CLONING**.**

Rule for the agents on node x:

1. Wait until all the smaller neighbours of x are clean or guarded.
2. Clone enough agents to clean the bigger neighbors of x and then send one agent per edge to clean each of the bigger neighbors. If all the smaller neighbors are guarded or clean and there are no bigger neighbors, terminate.

Similarly to theorem 7 we have:

Theorem 10. *During the cleaning process of Algorithm* CLEAN WITH VISIBILITY AND CLONING *the agents clean all nodes and a clean node will not be recontaminated.*

Regarding the complexity, first observe that as agents have power, they can clone new agents whenever they are needed, thus there are always enough agents to clean the network.

Theorem 11. *The cleaning of the entire network with Algorithm* CLEAN WITH VISIBILITY AND CLONING *requires* $\log n$ *time units,* $\frac{n}{2}$ *agents and* $n-1$ *moves. The time and move complexities are optimal.*

PROOF The bound on the time units may be computed similarly to the proof of theorem 8.

Regarding the number of agents observe that as all agents have to terminate, instead of counting how many agents were ever created, we count how many agents terminate. By the cleaning strategy, we know that the edges and nodes traversed by the agents form the broadcast tree; every node is visited by exact one agent. By Algorithm CLEAN WITH VISIBILITY AND CLONING, the agent on a node terminates only if the node does not have bigger neighbors. It is easy to see there are 2^{d-1} such nodes. Each of these 2^{d-1} nodes is visited by exactly one agent, which later terminates on it. Hence the total number of agents used is equal to $2^{d-1} = \frac{n}{2}$.

Finally, the computation of the number of moves trivially follows from the fact that the edges and nodes traversed by the agents form the broadcast tree; each edge in the broadcast tree is traversed by one agent only.

The time complexity is optimal because $\log n$ is the diameter of the hypercube, and the agents start from the same location. The number of moves is also optimal because all nodes must be visited. ∎

Synchronous Model with Cloning. We now examine the cleaning strategy in a synchronous environment where the agents have only cloning capability. Also in this case there is no need for a coordinator.

In this setting synchronicity could be exploited by using a strategy very similar to the one of Algorithm CLEAN WITH VISIBILITY but without needing the visibility assumption. Instead of waiting for all smaller neighbor(s) to become clean or guarded, the agents on a node wait for the appropriate time to move and clean the bigger neighbors.

Recall that $m(x)$ denotes the position of the most significant bit of x; in the synchronous model, the agents on x can move to the bigger neighbors when time $t = m(x)$ as they *implicitly* know that at this instant all the smaller neighbor(s) of x are clean or guarded.

At time 0, one agent is placed at node $(00...00)$; it cleans node $(00...00)$ and clones $d-1$ new agents; and then one agent per edge is sent to clean each of the d neighbors. At time i, an agent arrives at a node, it cleans the node and then executes the following algorithm.

Algorithm CLEAN WITH CLONING AND SYNCHRONICITY.

Rule for the agents on node x:

1. Wait until $t = m(x)$.
2. Clone enough agents to clean the bigger neighbors of x and then send one agent per edge to clean each of the bigger neighbors. If there are no bigger neighbors, terminate.

The correctness results follow from:

Lemma 6. *At time i, $0 \leq i \leq d$, of Algorithm* CLEAN WITH CLONING AND SYNCHRONICITY, *there is one agent on every node in C_i. This node clones other agents and cleans the bigger neighbors. At time d, all the agents on the nodes of C_d terminate.*

PROOF By Induction. Base case, at time $i = 0$, exactly one agent is placed on node $(00..00)$ by the strategy. Only node $(00..00)$ is in C_0, and by the strategy this agent clones and cleans its neighbours.
Assume the claim is true up to time i for $i \geq 0$. We show that at time $i + 1$, there is one agent on every node in C_{i+1} and this agent clones and either moves to a bigger neighbour (if any) or terminates. Let node x be any node in C_{i+1}. By property 7, exactly one smaller neighbor of x is in C_j, for some $j \leq i$. Let us call this smaller neighbor z. By induction hypothesis, at time j, there is one agent on z. Hence, by Algorithm CLEAN WITH CLONING AND SYNCHRONICITY, at time j, one agent is sent from z to x and has to stop at x up to time $t = i + 1$. At this time if $i + 1 \leq d - 1$ the agent clones enough new agents and cleans its neighbour. So the claim holds for time $i + 1$. If $i + 1 = d$ the agent is in C_d, i.e., is a leaf, thus it terminates. ∎

We now prove that our cleaning strategy is correct; i.e., that the network is clean and once a node has been cleaned, it will never be recontaminated.

Lemma 7. *At time i of Algorithm* CLEAN WITH CLONING AND SYNCHRONICITY, *all nodes in C_j, for $0 \leq j \leq i$, are clean.*

PROOF By induction. Base case: it is trivial for $i = 0$. Only node $(00..00)$ is in C_0 and at time 0 it is cleaned by the agent placed on it; the agent clones itself and sends the other agents to the bigger neighbours, so no recontamination may occur and at time 1 all the nodes in C_0 and C_1 are clean.
Assume the claim holds at time i, $i \geq 0$. Let us show that it is true at time $i + 1 \leq d$.
By lemma 6, at time $i+1$, every node in C_{i+1} is guarded by an agent. Let x be an arbitrary node of C_{i+1}. By property 7, one smaller neighbor of x is in C_j, $j \leq i$, let us call it y; and all other smaller neighbors of x, if any, are in C_{i+1} and guarded by agents at time $i + 1$. By induction hypothesis, at time i, nodes in C_j, $j \leq i$ are clean. Thus, y is clean. The only contaminated neighbors of x are the bigger neighbors but the agent on on x clones enough new agents, cleans its neighbours and moves from a guarded to a clean state together with its smaller neighbours in C_{i+1}. Finally observe that node recontamination can occur to nodes in C_{i+1} as all their neighbours are either clean or guarded and only move to a clean state at time $i + 2$ and remain in such a state. ∎
It follows from lemma 7 that:

Theorem 12. *During the cleaning process of Algorithm* CLEAN WITH CLONING AND SYNCHRONICITY *the agents clean all nodes and a clean node will not be recontaminated.*

Regarding the complexity we have the following:

Theorem 13. *The cleaning of the entire network with Algorithm* CLEAN WITH CLONING AND AND SYNCHRONICITY *requires $log\ n$ time units, $\frac{n}{2}$ agents and $n - 1$ moves. Time and move complexities are optimal.*

PROOF The bound on the time units comes from lemma 7, and that on the number of agents and moves is similar to the proof of theorem 11. ∎

4 Conclusion

In this paper we have considered the problem of searching for an intruder in a hypercube network. We have presented four different strategies that run in distinct models. Our results are summarized in the following table.

	Agents	**Time**	**Moves**
Coordinator	$O(\frac{n}{\sqrt{\log n}})$	$O(n \log n)$	$O(n \log n)$
Visibility	$n/2$	$\log n$	$O(n \log n)$
Visibility and Cloning	$n/2$	$\log n$	$n-1$
Synchronous with Cloning	$n/2$	$\log n$	$n-1$

Our first strategy uses the optimal number of agents and, as a consequence of our result, we have shown that $\Theta(\frac{n}{\sqrt{\log n}})$ is the search number for the hypercube in the classical model. The hypercube, like the tree (see [1]) is an example of a topology where "jumping" does not help reducing the number of agents.

In the subsequent strategies, our goal was to understand the power of additional assumptions, like visibility, cloning, and synchronicity. In this regard, we have observed that the visibility assumption allows to obtain an optimal time complexity; however, in our algorithm that exploits visibility, the number of required agents is not optimal any more. The major advantage of the visibility strategy however, is the fact that agents are autonomous and there is no need for a leader. Adding the cloning capability, we can reach also an optimal move complexity, the same result is obtained when cloning is combined with synchronicity.

Still open is the possibility of deploying an optimal number of agents without using a coordinator (i.e., in the visibility model), or to prove that it cannot be done.

Finally, let us observe that the need of whiteboards for communication among agents could be replaced by other assumptions. In the case of algorithm CLEAN, communication can be achieved even if there are no whiteboards, as long as an agent is initially designated as a leader and communication between agents is assumed to be *face-to face* (i.e., agents can communicate directly when they meet in a node). In all the other algorithms the whiteboards are never used for communication; in fact the agents act independently and they never need to exchange information. In these cases the whiteboards are only used to store local information (outgoing links, state of the node).

References

[1] L. Barrière and P. Flocchini and P. Fraignaud and N. Santoro. Capture of an intruder by mobile agents. *Proceedings 14-th ACM Symposium on Parallel Algorithms and Architectures*

(SPAA), Winnipeg, Manitoba, Canada, 200-209, 2002.

[2] L. Barrière and P. Fraignaud and N. Santoro and D.M. Thilikos. Searching is not jumping. *Proceedings 29th Workshop on Graph Theoretic Concepts in Computer Science (WG)*, Elspeet, the Netherlands, Springer Verlag, Lecture Notes in Computer Science 2880, 34-45, 2003.

[3] L.S. Chandran and T. Kavitha and C. R. Subramanian. Isoperimetric inequalities and the width parameters of graphs. *Proceedings Computing and Combinatorics, 9th Annual International Conference (COCOON)*, Big Sky, MT, USA, Lecture Notes in Computer Science 2697 Springer Verlag, 385-393, 2003.

[4] L.S. Chandran and T. Kavitha. The treewidth and pathwidth of Hypercubes. To appear in Discrete Mathematics 2005.

[5] J.A. Ellis and I.H. Sudborough and J.S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113: 50-79, 1994.

[6] P. Flocchini and F.L. Luccio and L.X. Song. Size Optimal Strategies for Capturing an Intruder in Mesh Networks. *Proceedings International Conference on Communications in Computing (CIC)*, Las Vegas, USA, 200-206, 2005.

[7] R.L. Graham and D.E. Knuth and O. Patashnik. Concrete Mathematics, Second Edition, Reading, Massachusetts, Addison-Wesley, 1994.

[8] L. M. Kirousis and C. H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47:205-218, 1986.

[9] A. Lapaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2): 224-245, 1993.

[10] N. Megiddo and S. Hakimi and M. Garey and D. Johnson and C. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35(1): 18-44, 1988.

[11] T. Parson. Pursuit-evasion problem on a graph. *Theory and applications in graphs*, Lecture Notes in Mathematics, Springer-Verlag, 426-441, 1976.

[12] T. Parson. The search number of a connected graph. *Proceedings of 9-nth Southeastern Conference on Combinatorics, Graph Theory and Computing*, Utilitas Mathematica, 549-554, 1978.

[13] S. Peng and M. Ko and C. Ho and T. Hsu and C. Tang. Graph searching on chordal graphs. *Algorithmica*, 27: 395-426, 2000.