Distributed Error Confinement[*]

Yossi Azar
azar@cs.tau.ac.il
Dept. of Computer Science
Tel Aviv University
Tel Aviv 69978
Israel

Shay Kutten
kutten@ie.technion.ac.il
Dept. of Industrial Engineering
The Technion
Haifa 32000
Israel

Boaz Patt-Shamir
boaz@eng.tau.ac.il
Dept. of Electrical Engineering
Tel Aviv University
Tel Aviv 69978
Israel

January 9, 2005

Abstract

We initiate the study of error confinement in distributed applications, where the goal is that only nodes that were directly hit by a fault may deviate from their correct external behavior, and only temporarily. The external behavior of all other nodes must remain impeccable, even though their internal state may be affected. Error confinement is impossible if an adversary is allowed to inflict arbitrary transient faults on the system, since the faults might completely wipe out input values. We introduce a new fault tolerance measure we call *agility*, which quantifies the strength of an algorithm that disseminates information against state corrupting faults.

We study the basic problem of broadcast, and propose algorithms that guarantee error confinement with optimal agility to within a constant factor. These algorithms can serve as building blocks in more general reactive systems. Previous results in exploring locality in reactive systems were not error confined, and relied on restrictive assumptions. Our results include a new technique that can be used to analyze the "cow path" problem.

1 Introduction

One of the key differences between centralized and distributed systems is that in distributed systems, faults may hit only a part of the system, whereas a centralized system is monolithic by definition. Consider transient faults: when the fault event is over, everything is functioning properly, but the *state* of the nodes hit by the fault may be wrong. The state of the system is not directly important, though: what matters is the *external behavior* of the system, which, of course, depends on the state. Faulty nodes may output wrong values, exposing the fault to the local users. However, while the damage at the nodes hit by the fault is unavoidable, the externally observable effect of a fault may reach much further: the system, which in most cases is designed to spread information, might actually amplify the effect of a fault by spreading harmful data across the system (see, e.g., the infamous crash of the Arpanet [28]).

[*] An extended abstract of this paper appeared in Proc. 22nd ACM Symp. on Principles of Distributed Computing, pages 33–42, July 2003.

Fault-resilient protocols traditionally deal with faults by allowing arbitrary behavior until recovery is complete (intuitively, declaring a temporary "state of emergency"). In this paper we devise systems that in addition to recovering from arbitrary transient faults, keep the faulty information masked from as many external users as possible, even during the recovery period.

Let us be a little more specific (see Section 2 for formal definitions). We consider a distributed system that consists of a collection of interconnected nodes, that collectively executes some reactive task. Abstractly, there is an "environment" (representing the users), that inputs values at nodes and reads outputs from nodes. The system functionality is specified as a predicate over the sequences of input and output values, which says what are the legal inputs and what are the allowed outputs for each legal input. We consider transient faults, i.e., faults that eventually leave the system. This is modeled by assuming that a fault may hit a set of nodes by arbitrarily modifying their state. Note that a fault is an abstraction of a "batch" of faults that may hit many nodes simultaneously.

The main property our protocols enjoy is *error confinement*. Intuitively, a system is said to have the error confinement property if in any execution, the observable input and output values at all nods meet the specification, except possibly nodes that were directly hit by a fault. This is the best one can ask for: a local fault can trivially violate the specification at a faulty node, say by changing the value of a local output variable. Moreover, we note that this is often good enough. For example, if the task is routing in a large network and a few nodes are hit, then typically, most routes will not be affected. In addition, error confinement allows for layered design of resilient protocols: a high level error-confined system can be built on top of a low-level error-confined system (in fact, we use this methodology in our paper).

To be concrete, in this paper we study the basic problem of *broadcast*, where a value is input at one of the nodes and the task is that eventually, all other nodes will output that value exactly. The problem of broadcast is interesting in our context for two reasons. First, the essence of broadcast is dissemination of information, in apparent contrast to the idea of error confinement, since the operation of the protocol might contaminate non-faulty nodes with bad information. Second, broadcast is, in some sense, a complete problem for reactive tasks: by broadcasting all input values, all nodes can know all inputs, and each node can compute the output locally for any reactive task.

One difficulty with the concept of error confinement is that it is impossible to attain in general, because if a value is input at a node, and an error hits that node before it sent out any message, then the input value may have no trace whatsoever in the system, making perfect recovery impossible. On the other hand, if the fault hits only a minority of the nodes after the input value is already safely mirrored in all nodes, recovery is possible [18]. In this paper we consider the broadcast protocol that starts when an input value resides at one node (the source) and ends when the input value is mirrored everywhere. Thus, our protocol covers the critical interval that starts when recovery even from a single-node fault is impossible, and ends when it is possible to recover from any fault that hits as many as $\lfloor \frac{n-1}{2} \rfloor$ nodes. A good protocol should increase the fault resilience as quickly as possible during the dissemination interval. We capture this intuition by introducing a new measure we call *agility*, which quantifies the resilience of reactive algorithms as a function of time. Intuitively, the algorithm must confine faults if they hit only a minority of the nodes in a ball around the origin of the input value. An algorithm is said to be more agile if that ball grows more quickly, i.e., the agility of the algorithm measures *how quickly do we lift the restriction on the faults.* For example, an algorithm that cannot recover from a corrupted source has agility 0. On the other hand, an algorithm that can recover, for any time t from a fault that corrupts only a minority of the nodes in distance at most t from the source is said to have agility 1 (assuming that messages travel one distance unit per time unit). As we show, agility cannot reach 1 in error confined protocols, i.e., the process of establishing correct mirrors is inherently slower than the propagation speed of messages. If an an algorithm can recover at any time t from a fault that corrupts only a minority of the nodes in the ball of, say, radius $\frac{t}{f(n)}$ around the source then its agility is $\frac{1}{f(n)}$. It turns out that the optimal agility algorithm is not greedy. That is, a policy of increasing the number of mirrors (and thus increasing the number of faults the algorithm can withstand in the next moment) is not an optimal policy for the agility over all the run of the algorithm. Instead, in the optimal algorithm, even nodes that have received the broadcast already are not used immediately as mirrors. Instead, the algorithm waits until a large number of additional nodes receives the broadcast, and only then declares them as mirrors.

Our contribution. The conceptual contributions of this paper are the formalization of the notions of error confinement and agility. (The concept of agility is applicable also for protocols which are not error-confined.) The technical contribution of the paper is a study of error-confined protocols for the basic primitive of broadcast, including an algorithm and a lower bound on the slowdown speed of any such algorithm. Specifically, we prove that error confinement necessarily entails a slowdown of factor 2 for broadcast, and we present an algorithm with slowdown $6 + 4\sqrt{2} \approx 11.7$.

The algorithm uses a novel tool called *ball core*, which is natural in our context. For algorithms using ball cores, we present matching lower and upper bounds on the agility of any broadcast algorithm. We remark that the analysis of ball cores may be of independent interest, as it includes a new technique that can be used to analyze the cow path problem [23, **?**].

In addition to confining faults, our protocol is self stabilizing, i.e., regardless of the extent of the fault, the system will eventually reach a legal state. In other words, recovery of the input is guaranteed only if the faults satisfy to a certain limitation, but the system will stabilize in any case (possibly as if another value was input).

Related work. Our approach in modeling faults is similar to the one in [21, 8]. The model of state-corrupting faults is implicit in the work of Dijkstra about *self stabilization* [11]: put in our terminology, a system is called self-stabilizing if after an arbitrary state-corrupting fault occurs (possibly hitting all nodes), eventually the system starts behaving correctly. Some general algorithmic solutions for making a system self-stabilizing appear in [16, 4, 6, 12], based on the paradigm of *reset* [7]: when an error is detected, a global reset action is invoked, whose effect is to impose a correct state on the system.

Many fault-resilient protocols allow incorrect behavior until recovery is complete. Some protocols limit the *kind* of incorrect behavior permitted [12]. In [17, 3, 18, 14], the question is how to reduce the effect of faults in *time*, i.e., have short recovery time if the number of faults is small. By comparison, in error confinement, the goal is to reduce the effect of faults in *space* (i.e., portion of the network), i.e., the set of eventually affected nodes should be related to the set of nodes directly hit by a fault. In [10], a system is called *snap-stabilizing* if its behavior stabilizes to its specification in 0 time. Clearly, snap stabilization is possible only for the limited class of tasks that allow a faulty node to be considered externally correct even at the time of the fault (broadcast does not satisfy this requirement).

In [3], every faulty node can detect itself faulty, since it is assumed that the nature of the faults was probabilistic. While error confinement is not explicitly considered in [3], it is easily attainable: since a node can detect itself as faulty, it can avoid sending wrong information. A similar idea is used in [27], where a fault can be detected by a neighbor. Obviously, these fault models are much weaker than the one we use.

In [18, 17], the goal is to maintain the correctness of data in the face of faults that occur after the data is replicated; the replication process is not in the scope of these papers. Informally, the task there is a "special" kind of a consensus, in which each non-faulty node votes its local replica value (faulty nodes may vote a wrong value). That consensus is special because it runs in a self stabilizing model, and because its output stabilization time is proportional to the number of faults.

We stress that all the above papers allow for correct nodes to exhibit faulty behavior before stabilization. A simple type of error confinement is achieved in [22], where it is assumed that there are t correct sources to begin with, and at most $t-1$ nodes may be faulty; the faults considered in [22] are Byzantine, rather than transient. On one hand, no stabilization can be guaranteed in this case, but on the other hand, it is easy to ensure error confinement by allowing each node to make an output only after it gets identical values from t distinct nodes.

Error confinement is an important aspect in fault-tolerant software systems. See, e.g., [20, 25] and references therein for the software engineering perspective on fault confinement.

Organization of this paper. The remainder of this paper is organized as follows. In Section 2 we formally define the model and the concepts of error confinement and algorithm agility. In Section 3 we develop an error-confined algorithm for the broadcast problem in the synchronous model and analyze its agility. The analysis of ball cores (which is related to the cow path problem) is presented in Section 3.4. We conclude with some discussion in Section 4.

2 Basic Concepts

In this section we define the model of computation, the broadcast task, and the key concept of algorithm agility.

2.1 A Model for Error Confinement

General parameters. The system is modeled as a fixed undirected connected graph $G = (V, E)$, where nodes represent processors and edges represent bi-directional communication links. We denote $|V| = n$. The distance between two nodes $u, v \in V$, denoted $\mathsf{dist}(u, v)$, is the minimal number of edges in a path connecting them. Given a node $v \in V$, we denote $\mathsf{ball}_v(r) = \{u \in V \mid \mathsf{dist}(v, u) \leq r\}$, and call it the *ball* of radius r around v. The diameter of a graph is denoted diam, and is defined to be the minimal r such that for all $v \in V$, $\mathsf{ball}_v(r) = V$.

- An *action* is associated with (a.k.a. occurs in) a *node.* An action may be either *external* or *internal.* An external action is either an *input* or an *output* action. Nodes may also be called “processors,” or “sites.”
- A *behavior* is a sequence of external actions. Given a node v and a behavior β, β_v is the *local behavior* of v, i.e., the subsequence of β that consists only of actions that occur in v. Consecutive external actions are idempotent, i.e., making a single output or input action is equivalent to repeating that action any positive number of times.
- A *task* is specified by a set of behaviors, called *legal behaviors* for the task.
- A *protocol* is a specification of *states* and *transitions.* A state is a vector of *local states*, one local state for each node. Transitions are triples denoted $s \xrightarrow{a} s'$, where s and s' are states, and a is an action (not necessarily an external one). An action a is said to be *enabled* in state s if $s \xrightarrow{a} s'$ is a transition for some state s'. Input actions are enabled in all states. A non-empty subset of the states is designated as *initial states.*
- An *execution* of protocol P is an infinite sequence $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$, where s_0 is an initial state of P, and $s_{i-1} \xrightarrow{a_i} s_i$ is a transition of P for all i. In a *timed* execution, each action is annotated with its time of occurrence.
- Given an execution e, its corresponding behavior, denoted $\beta(e)$, is the sequence of all external actions in e.
- A protocol P *implements* task Π if its set of behaviors is a subset of the legal behaviors of Π.

Figure 1: *Actions, behaviors, states, executions etc.: IO Automata formalism.*

Error confinement. To define error confinement formally, we use IO Automata as our underlying formalism [21]. The standard definitions are summarized in Figure 1. In this paper we consider a single type of faults, called *state corrupting*, that abstracts all transient faults. Such faults are formally defined as follows.

Definition 2.1 *A* state corrupting fault *is an action that alters the state arbitrarily in some subset of nodes. The nodes whose local state was altered are called* faulty*.*

In our model, there is at most one fault in an execution, which can span a set of nodes. (This means that we assume that faults are sufficiently separated so as to allow complete system stabilization between them.)

The new concept we propose is the following.

Definition 2.2 *A protocol P is said to be an* error-confined *protocol for task* Π *if for any execution with behavior β (possibly containing a fault) there exists a legal behavior β' of* Π *such that*

(1) For each non-faulty node v, $\beta_v = \beta'_v$.

(2) For each faulty node v, there exists a suffix $\overline{\beta_v}$ of β_v and a suffix $\overline{\beta'_v}$ of β'_v such that $\overline{\beta_v} = \overline{\beta'_v}$.

The output stabilization time *of a faulty node v is the time duration of the prefix of β_v that is not included in $\overline{\beta_v}$.*

The main point in the definition above is that the behavior of non-faulty nodes must be exactly as in the specification: only faulty nodes may have some period (immediately following the fault) in which their behavior does not agree with the specification.

Broadcast. In this paper we study the problem of disseminating a value from a given source node. Formally, the broadcast task is defined as follows.

Broadcast (BCAST)

Input actions: $\mathtt{inp}_s(b)$, done at node $s \in V$, and b in some set D. The node s is called *source.*

Output actions: $\mathtt{outp}(b)$, required at all nodes $v \in V$, where $b \in D \cup \{\perp\}$.

Legal behaviors: There is at most one $\mathtt{inp}_s$ action. Each node v outputs $\mathtt{outp}(\perp)$ in each step up to some point, and then it outputs $\mathtt{outp}(b)$ in each step, where b is the value input by the `inp` action.

For ease of exposition, we abuse notation slightly and use special input and output registers called `mirror`: an input action is equivalent to assigning a value to the $\mathtt{mirror}_s$ register at the source s, and similarly the output action at node v just reads the value of the local $\mathtt{mirror}_v$ value. (We use the convention that variables are subscripted by their node name.) We assume that the input occurs at some time t_0 that is known to all the nodes. Informally, the idea is that in an interactive task each source receives an infinite stream of inputs- one input per time unit. In this paper we pick one of these time units (so the time is known) and deal with distributing the value input at that time unit (If no input occurs at that time unit then the algorithm distributes a special nil value.)

Error confined broadcast means that if any non-faulty node outputs a value $a \neq \perp$, then all non-faulty nodes may output only a (or $\perp$), and all nodes must output a eventually.

Agility. As mentioned above, there is no way to maintain error confinement in the face of an arbitrary state-corrupting fault: the fault may hit the source immediately after the input action, leaving no trace of the original input value. However, faults can be overcome if they arrive later, since the source could have communicated the input to some other nodes in the meantime. It is impossible to design an algorithm that will ensure replication to more nodes than those in a certain distance (a *ball* around the source) that depends on the time. If a fault hits the majority of nodes in this ball, ensuring recovery is impossible. The notion of α-constrained environment formalizes this idea.

Definition 2.3 *An environment is called* $\alpha(t)$-constrained *for some function* $\alpha(t)$ *and a given system topology if the following condition holds. Suppose that input is made at node* s *at time* t_0*, and that a fault occurs at time* t_f*. Then the number of nodes hit by the fault is less than* $\frac{1}{2}|\mathsf{ball}_s(\alpha(t_f - t_0))|$.

The propagation of information in the system is physically limited to within a dynamically growing ball centered at the source, and hence no algorithm can recover inputs if that ball is corrupted. Our definition restricts the faults inside a ball whose radius growth is bounded by the function $\alpha(t)$.

Definition 2.4 *An algorithm for the broadcast problem* agility $\alpha(t)$ *if it has the error-confinement property for all* $\alpha(t)$*-constrained environments. An algorithm is said to have agility* c *for a constant* c *if it has* $\alpha(t)$ *agility for* $\alpha(t) = c \cdot t$ *for all* $t \geq 0$.

Note that while our definition is for error-confinement, it generalizes for any type of fault resilience. The agility of an algorithm, intuitively, defines the maximum rate in which the ball grows that still allows the algorithm to be correct.

Synchronous computations. The *synchronous network model* is a rather common one (see, e.g. [1, 5]). In this model, time proceeds in steps, where in each step (called *round* of computation), all nodes first read the state of their neighbors, and then set their own state. This model abstracts the underlying mechanism whose job is to make the state available at neighbors, as well as synchronize their progress. Note that in the synchronous model, states change at discrete steps. When we say "at time t," the interpretation is "in the state between the end of step t and the start of step $t+1$."

3 Broadcast with Error Confinement

In this section we develop an algorithm for BCAST with error confinement in the synchronous model. We start, in Section 3.1, with an algorithm that works only if the source is never faulty. Using that algorithm as a subroutine, we present, in Section 3.3, our final algorithm, which works even if the source is faulty. The latter algorithm uses a novel concept we call *ball cores*, analyzed in Section 3.4.

Our algorithms slow down the propagation of information. In Section 3.2 we prove a lower bound, that says that any algorithm for broadcast must slow down the output by at least a factor of 2, even if the source is guaranteed to never be hit by a fault.

3.1 A building block: Broadcast with a correct source

In this subsection we solve BCAST under the assumption that the source is correct. This primitive is useful since it has the following partial error confinement property unconditionally (whether the source was hit by a fault or not): if the algorithm outputs a value at a non-faulty node v, then the output value is *authentic*, in the sense that it was indeed communicated by the source. This is not full error confinement: the output values, although authentic, may be faulty if the source is faulty.

To solve BCAST with a correct source, we start with the problem of distance computation.

Single source distance computation (SSD)

Input actions: none.

Output actions: d, where $d \in \{0, 1, \ldots, N\} \cup \{\perp\}$ for some large integer N.

Legal behaviors: Each node v outputs $\perp$ in each step up to some point, and then it outputs $\mathsf{dist}(s, v)$ in each subsequent step, where s is the *source* node.

Without the requirement for error confinement, the Bellman-Ford algorithm solves SSD even in the face of state-corrupting faults (see, e.g., [5]). Informally, the algorithm works as follows. In non-source nodes, the initial value of the output variable $\mathtt{dist}_v$ is $\perp$, and in each step, the node sets its value to be one plus the minimum of the distance variables of its neighbors (where $\perp$ is treated as infinity). The source node sets its output variable to 0 in each step.

However, the Bellman-Ford algorithm is not error-confined: For example, if a fault causes a non-source node to set its distance to 0, its neighbors will set their outputs to 1 in the following round, violating error confinement. Nevertheless, a simple extension makes Bellman-Ford error-confined. As

```
State at node $v \neq s$:
    $\texttt{dist}_v$: the distance variable to be output, initially $\bot$
    $\texttt{cand\_dist}_v$: an internal estimate of the distance, initially $\bot$
    $\texttt{count}_v$: counter, initially 0

Code at node $v \neq s$:
    if $\texttt{cand\_dist}_v \neq 1 + \min\{\texttt{cand\_dist}_u \mid u$ is a neighbor of $v\}$        $\bot$ is treated as infinity
        then                                                                                             change detected
            $\texttt{cand\_dist}_v \leftarrow 1 + \min\{\texttt{cand\_dist}_u \mid u$ is a neighbor of $v\}$    update variable...
            $\texttt{count}_v \leftarrow 0$                                                              ... and reset counter
        else $\texttt{count}_v \leftarrow \min(\texttt{count}_v + 1, \texttt{cand\_dist}_v)$             increment counter, but keep it bounded
    if $\texttt{count}_v \geq \texttt{cand\_dist}_v$
        then $\texttt{dist}_v \leftarrow \texttt{cand\_dist}_v$                                          expose value in output register

Code at node $s$:
    $\texttt{dist}_s \leftarrow 0$
```

Figure 2: *Algorithm A: Single source distance computation with confined errors.*

we prove below, it turns out that if the distance variable value is d, and it has not changed for at least d time units, then the distance of the node from the source is indeed d. This property gives rise to Algorithm A, presented formally in Figure 2.

We now prove the error confinement property of Algorithm A. Let us assume without loss of generality that the fault occurs at time t_f, and that at that point, the state variables have arbitrary values. We have the following two properties.

Lemma 3.1 *At any time $t \geq t_f$, for any node v, we have $\texttt{cand\_dist}_v \geq \min(\mathsf{dist}(s,v), t - t_f)$.*

Proof: We start by noting that the lemma holds trivially for the source node, since by the code $\texttt{cand\_dist}_v = 0$ always, and since $\mathsf{dist}(s,s) = 0$. To prove the lemma for non-source nodes, we use induction on time. For $t = t_f$ the lemma holds trivially since $\texttt{cand\_dist}_v \geq 0$ always and $t \geq t_f$ by assumption. For the inductive step, assume that the lemma holds for all nodes at time $t \geq t_f$, and consider time $t+1$. Let $v \neq s$ be any non-source node. We proceed by cases. If $(t+1) - t_f \leq \mathsf{dist}(s,v)$, we need to prove that at time $t+1$, $\texttt{cand\_dist}_v \geq (t+1) - t_f$. By induction we have that at time t, for all neighbors u of v, $\texttt{cand\_dist}_u \geq \min(t - t_f, \mathsf{dist}(s,u) = t - t_f$, since $\mathsf{dist}(s,u) \geq \mathsf{dist}(s,v) - 1 \geq (t + 1 - t_f) - 1$. It follows that at time $t+1$, v assigns to $\texttt{cand\_dist}_u$ a value which is at least

$$\begin{aligned} \texttt{cand\_dist}_v &\leftarrow 1 + \min\{\texttt{cand\_dist}_u \mid u \text{ is a neighbor of } v\} \\ &\geq 1 + (t - t_f) = (t+1) - t_f \ , \end{aligned}$$

and we are done in this case. Suppose now that $(t+1) - t_f > \mathsf{dist}(s,v)$. We need to prove that at time $t+1$, $\texttt{cand\_dist}_v \geq \mathsf{dist}(s,v)$. Consider a neighbor u_0 of v with minimal distance from s. Since $\mathsf{dist}(s,u_0) = \mathsf{dist}(s,v) - 1$ we have in this case that $t - t_f > \mathsf{dist}(s,u_0)$. Moreover, by the induction hypothesis, we have that $\texttt{cand\_dist}_u \geq \mathsf{dist}(s,u_0)$ for all neighbors u of v. It follows that at time

$t - t_f + 1$, v assigns to $\texttt{cand\_dist}_v$ a value of at least $\mathsf{dist}(s, u_0) + 1 = \mathsf{dist}(s, v)$, and we are done in this case too. ▮

Lemma 3.2 *Let v be any node, and let d be such that $\mathsf{dist}(s, v) \leq d$. Then at any time $t \geq t_f + d$, we have that $\texttt{cand\_dist}_v \leq d$.*

Proof: By induction on the distance $\mathsf{dist}(s, v)$. If $\mathsf{dist}(s, v)$=0, then $v = s$ and the claim follows trivially from the code. Assume that the claim is true for all nodes at distance d at all time steps $t \geq t_f + d$, and consider a node v with $\mathsf{dist}(s, v) = d + 1$. Let u_0 be a neighbor of v with $\mathsf{dist}(s, u_0) = d$. By the induction hypothesis, at time $t_f + d$ and onward, we have the $\texttt{cand\_dist}_{u_0} \leq d$. It follows from the code that at time $t_f + d + 1$ and onward, $\texttt{cand\_dist}_v \leq d + 1$, as required. ▮

Using the lemmas above, we now analyze the Single Source Distance calculation time.

Theorem 3.3 *Algorithm A solves SSD with confined errors and output stabilization time* $2 \cdot \mathsf{diam}$.

Proof: We first prove stabilization. By Lemmas 3.1 and 3.2, we have $\texttt{cand\_dist}_v = \mathsf{dist}(s, v)$ for any time $t \geq t_f + \mathsf{diam}$, at any node v. Hence, $\texttt{count}$ is never reset to 0 after time diam, and therefore, by time $t_f + 2 \cdot \mathsf{diam}$ we have $\texttt{count}_v \geq \mathsf{dist}(s, v) \geq \texttt{cand\_dist}_v$. It follows that by time $t_f + 2 \cdot \mathsf{diam}$, all nodes have $\texttt{dist}_v = \mathsf{dist}(s, v)$, and the system stabilizes as required. Next, we show error confinement. We need to show that if when a node outputs a value (i.e., it sets its $\texttt{dist}$ variable), this value is correct or else that node is faulty. Let v be any node, and suppose that v changes the value of its $\texttt{dist}_v$ variable at time t. There are two cases to consider. If the value of $\texttt{dist}_v$ is changed less than $\texttt{cand\_dist}_v$ time units since the last time the $\texttt{cand\_dist}_v$ variable was changed, then clearly the $\texttt{count}_v$ variable does not have its intended semantics, which necessarily means that node v is faulty and error confinement is trivially satisfied. Otherwise, suppose that $\texttt{dist}_v$ is set to d after at least d time units in which $\texttt{cand\_dist} = d$. By Lemma 3.1 we have that $\mathsf{dist}(v, s) \geq d$, and by Lemma 3.2 $\mathsf{dist}(v, s) \leq d$, and hence $\texttt{dist}_v$ is set to $\mathsf{dist}(s, v)$. The theorem follows. ▮

We remark that Theorem 3.3 holds for a fault that hits any number of nodes at any time, simply because there is no input value.

We now extend Algorithm A to solve the BCAST task. This is done simply by "piggy-backing" the broadcast value on the distance value, once it is input. The broadcast value becomes externally visible only when the $\texttt{dist}$ variable becomes visible in Algorithm A. The algorithm for broadcast with error confinement, called Algorithm B, is formally presented in Figure 3 for non-source nodes. For the source node s, we have that the $\texttt{inp}_s(b)$ action results in assigning $\texttt{mirror}_s \leftarrow b$, and also the source keeps setting $\texttt{cand\_dist}_s \leftarrow 0$ and $\texttt{dist}_s \leftarrow 0$ in each subsequent step.

Theorem 3.4 *If the source node is non-faulty, then Algorithm B solves BCAST with confined errors, and output stabilization time* $2 \cdot \mathsf{diam}$.

Proof: Consider a non-faulty node v, and suppose that it assigns a value to $\texttt{mirror}_v$ at time t. We show that this value is correct. Suppose that at time t, $\texttt{cand\_dist}_v = d$. Since v is non-faulty, we have by the code that at time t, $\texttt{count}_v = d$, and that there were at least d time units during which $\texttt{cand\_dist}_v$ did not change.

We claim that $d = \mathsf{dist}(s, v)$. First note that if $d > \mathsf{dist}(s, v)$, then by Lemma 3.2, by time t we

```
State at node $v \neq s$:
    $\mathtt{mirror}_v$: the broadcast value to be output, initially $\bot$
    $\mathtt{cand\_mirror}_v$: an internal estimate of the output value, initially $\bot$
    $\mathtt{cand\_dist}_v$: an internal estimate of the distance, initially $\bot$
    $\mathtt{count}_v$: counter, initially 0

Code at node $v \neq s$:
    Let $u_0$ be the neighbor of $v$ with $\mathtt{cand\_dist}_{u_0} = \min\{\mathtt{cand\_dist}_u \mid u \text{ is a neighbor of } v\}$
    if $(\mathtt{cand\_dist}_v \neq 1 + \mathtt{cand\_dist}_{u_0})$ or $(\mathtt{cand\_mirror}_v \neq \mathtt{cand\_mirror}_{u_0})$
        then
            $\mathtt{cand\_dist}_v \leftarrow 1 + \mathtt{cand\_dist}_{u_0}$
            $\mathtt{cand\_mirror}_v \leftarrow \mathtt{cand\_mirror}_{u_0}$
            $\mathtt{count}_v \leftarrow 0$
        else $\mathtt{count}_v \leftarrow \min(\mathtt{count}_v + 1, \mathtt{cand\_dist}_v)$
    if $\mathtt{count}_v \geq \mathtt{cand\_dist}_v$                    all is well, distance and mirror are correct
        then $\mathtt{mirror}_v \leftarrow \mathtt{cand\_mirror}_v$
```

Figure 3: *Algorithm B. Broadcast with confined errors assuming the source is non-faulty.*

have that $\mathtt{cand\_dist}_v = \mathsf{dist}(s, v) < d$, contradicting the assumption that $d > \mathsf{dist}(s, v)$. So it must be the case that $d \leq \mathsf{dist}(s, v)$. Suppose for contradiction that $d < \mathsf{dist}(s, v)$. We show that $\mathtt{count}_v$ must have been reset by time t in this case. We say that a node v *consistently depends* on node u in a given state if $\mathtt{cand\_dist}_v = \mathtt{cand\_dist}_u + 1$ and $\mathtt{cand\_mirror}_v = \mathtt{cand\_mirror}_u$. Nodes $v_0, v_1 \ldots, v_k$ are called a *consistent dependency chain* of v_0 in a given state if v_i consistently depends on v_{i+1} for all $0 \leq i < k$ in that state. Note that if the maximal consistent dependency chain of a node at some state is of length 0, then by code, that node will set $\mathtt{count}_v \leftarrow 0$ in the next step. Now, consider the maximal consistent dependency chain of v such that $\mathtt{cand\_dist}_x = d$ for every node in the chain. There exists one chain for each time step $t-1, t-2, \ldots, t-d$. We claim that at least one of these chains is of length 0, which contradicts the assumption that $\mathtt{count}_v$ was not reset during this time interval. To see that, first note that the length of the chain at time τ is exactly $d - \mathtt{cand\_dist}_{u(\tau)}$, where $u(\tau)$ is the last node in the chain of v at time τ. Since $d < \mathsf{dist}(s, v)$ by assumption, it follows from the triangle inequality that $\mathtt{cand\_dist}_{u(\tau)} < \mathsf{dist}(s, u(\tau))$, and hence, by Lemma 3.1 we have that at time $t - d + i$ the length of any consistent dependency chain of v is at most $d - i$, and hence there will exist a zero-length chain by time $t - 1$, as required. The output stabilization time follows directly from the fact that after diam time, all $\mathtt{cand\_dist}$ and $\mathtt{cand\_mirror}$ variables have the correct values, which in turn follows from Lemmas 3.1 and 3.2. ▮

3.2 Error confinement implies slowdown

Clearly, under Algorithm B, a node v outputs a value after $2 \cdot \mathsf{dist}(s, v)$ time units, even if there are no faults: twice the necessary minimum. The following theorem shows that this slowdown is inherent to error confinement, even if the source is guaranteed to be always correct. Furthermore, the slowdown must occur even in fault-free executions.

Theorem 3.5 *Let X be an algorithm solving BCAST with error-confinement if the source s is correct. Then for any non-faulty node v, the time in which v outputs a value is at least $2 \cdot \mathsf{dist}(s, v)$ steps after the input at s, even if there are no faults.*

Proof: Consider a line graph, where nodes are numbered $0, 1, 2, \ldots$, and let the source be node 0. Consider any node i. We compare two executions of X: in execution e_0, no input is ever made at the source, and in execution e_1, a value 1 is input at s at some time t_0. Let t_1 be the first time in which the execution of node v differs between e_0 and e_1. Obviously, $t_1 \geq t_0 + i$, since the first difference between e_0 and e_1 occurs at time t_0 at distance i from node i. To prove the theorem we claim that algorithm X cannot output a value at i before time $t_1 + i$ in e_1. This is shown by contradiction: Suppose that i outputs a value at time $t_2 < t_1 + i$. We define another execution e' as follows. Up to and excluding time t_1, e' is identical to e_0. At time t_1, two events occur at e': First, an input of value 0 is made at s; and second, a fault changes the states of the nodes number $i - 1, \ldots, i - (t_2 - t_1)$ to be the same as in e_1. Note that the source is non-faulty because $i > t_2 - t_1$. Also note that node i is non-faulty. It is immediate to verify by induction that the execution of nodes $i, i - 1, \ldots, i - (t_2 - t_1) + j$ is identical in e_1 and e' in steps $t_1, \ldots, t_1 + j$ since each of these nodes cannot distinguish e' from e_1 at these times. It therefore follows that node i will output value 1 in e', a contradiction to the error-confinement property that requires all outputs at non-faulty nodes to be the same as the input value. ▮

3.3 General Error-Confined Broadcast

We now present the main algorithmic contribution of the paper. This algorithm tolerates a faulty source, under the assumption that faults may not corrupt the state of a majority of the nodes in $\mathsf{ball}_s(\alpha(t))$ at time t, for some function $\alpha(t)$ we specify later. The basic idea is to use bootstrapping: While algorithm B uses only the original source node (and had agility zero), the algorithm we present in this section maintains a dynamically growing set of nodes which collectively function as the data source. This set, called the *core nodes*, is denoted by $\mathsf{core}(t)$, where t is the time index. To start the bootstrapping, $\mathsf{core}(0) = \{s\}$, where s is the original source node. Each node in the core set broadcasts (using Algorithm B as a subroutine) what it believes to be the true value input at s at time 0. Assuming that no fault ever directly corrupts the majority of the current core, the algorithm ensures that *always*, the majority of values in the core set is correct.

The core grows inductively: A node may join the core if it has "sufficient evidence" to determine that the value it is about to start broadcasting is correct. "Sufficient evidence" is interpreted as a majority of the values broadcast by a complete core set. This is sufficient since faults may corrupt only a minority of the core nodes by assumption. Thus, we need to specify how to select the next core in a way that will lift these constraints on the adversary as fast as possible. For now, let us use an abstract core function; we propose a specific function in Section 3.4.

For the subtask of correctly collecting the above "sufficient evidence" values, the algorithm uses Algorithm B as a building block (that's why the fact that Algorithm B is error confined is crucial). This leaves us with the task to design the algorithm in such a way that the assumption on the constraints of the faults is minimal. Specifically, consider the algorithm (with a parametric $\mathsf{core}(t)$ function) specified in Figure 4. The only thing in which the source node differs from other nodes is that its $\texttt{mirror}_v$ value is assigned by the environment. Denoting the time of occurrence of the input event by 0, we have that $\mathsf{core}(1) = \{s\}$.

```
State at a node $v$:
    $\mathtt{mirror}_v$: the broadcast value to be output, initially $\perp$

Code for a node $v$, executed in each time step $t$:
    Receive broadcasts from nodes (if any) using Algorithm B.
    if Algorithm B locally outputs values from all nodes in $\mathsf{core}(t-1)$
        then set $\mathtt{mirror}_v$ value to their majority value (otherwise, $\mathtt{mirror}_v$ retains its previous value).
    if $v \in \mathsf{core}(t)$
        then broadcast $\mathtt{mirror}_v$ using Algorithm B.
```

Figure 4: *Algorithm Boot: Broadcast with confined errors. The algorithm uses a function* core *that must satisfy the feasibility condition (see text).*

The algorithm works for any $\mathsf{core}(t)$ specification, so long as the following condition is satisfied for every node v and time step t:

Feasibility Condition: If $v \in \mathsf{core}(t) \setminus \mathsf{core}(t-1)$, then by time t node v received broadcast values $\mathtt{mirror}_u$ from all nodes $u \in \mathsf{core}(t-1)$.

Theorem 3.6 *Let* $\mathsf{core}(t)$ *be a function satisfying the feasibility condition, and assume* $\mathsf{core}(t) \supseteq \mathsf{ball}_s(\alpha(t))$ *for some function* $\alpha(t)$*. Then Algorithm Boot is an error-confined algorithm for broadcast with agility at least* $\min\left\{\frac{\alpha(t)}{t} \mid t \geq 1\right\}$ *and output stabilization time of* $2 \cdot \mathsf{diam} + \min\{t \mid \mathsf{core}(t) = V\}$.

Proof: We first prove, by induction on time, that if node v is non-faulty, then it will never set its $\mathtt{mirror}_v$ variable to a wrong value. The base case is $t \leq t_f$, where t_f is the time of the fault (possibly, $t_f = \infty$). In this case, the claim follows from the trivial fact that the system is correct before the fault. For the inductive step, suppose that a non-faulty node v sets its $\mathtt{mirror}_v$ value at time $t > t_f$. By Theorem 3.4, Algorithm B guarantees that the local outputs at v of the broadcasted values from nodes in $\mathsf{core}(t-1)$ are authentic. Since less than $\frac{1}{2}|\mathsf{core}(t_f)|$ nodes are faulty by assumption that $|\mathsf{core}(t)| \geq |\mathsf{ball}_s(\alpha(t))|$ and since all non-faulty nodes have correct $\mathtt{mirror}$ values by the induction hypothesis, and since all these values will arrive at v by the second part of the feasibility condition, v will set its $\mathtt{mirror}_v$ variable to the correct value. This proves the error confinement property.

The output stabilization time bound follows by the fact that non-faulty nodes broadcast the correct value and from Theorem 3.4. ▮

3.4 Ball Core Functions

Algorithm Boot uses an abstract feasible core function. In this section we focus on special kind of cores we call *ball cores.* Ball cores match our definition for constrained environments: They are defined by $\mathsf{core}(t) = \mathsf{ball}_s(R(t))$ for some $R(t)$ called the *radius* of the core at time t. By definition, the agility of a the Boot algorithm when using ball cores is $\min\{R(t)/t \mid t > 0\}$. In the remainder of this section, we show what ball core function admits the best possible agility.

So fix a ball core function core. Let $\{R_i\}$ denote the sequence of all *distinct* radii of core in increasing order, i.e., $R_i < R_{i+1}$ for all i. Let T_i denote the first time that $\mathsf{core}(t) = \mathsf{ball}_s(R_i)$. See Figure 5. To simplify exposition, and motivated by Theorem 3.5, from now on we normalize the time scale by a factor of 2, which means that the time between the start of Algorithm B at any node v and the time another node u has an authenticated value of $\mathtt{mirror}_v$ is $\mathsf{dist}(v, u)$ if no faults occur.

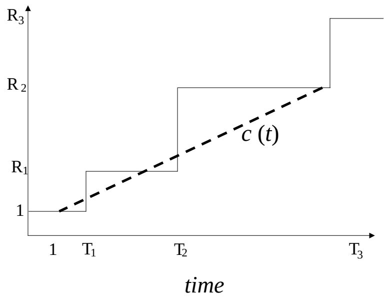

Figure 5: *The agility is the rate the constraint on the fault is released to allow more faults*

The following lemma establishes a strong connection between R_i and T_i (see Figure 6).

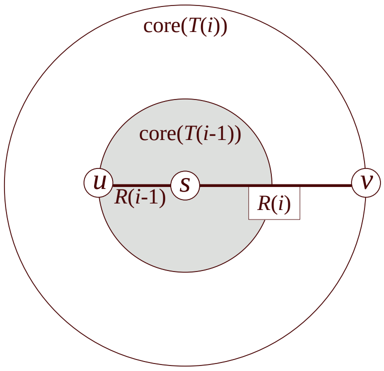

Figure 6: *The feasibility condition implies that node v cannot join the core before $R_i + R_{i+1}$ time units have passed since node u joined the core.*

Lemma 3.7 *Let* core *be a feasible ball core function. Then for all $i > 0$, we have*

$$T_i \geq R_i + 2\sum_{j=1}^{i-1} R_j \ .$$

Moreover, the best agility for the given radii sequence is attained when equality holds.

Proof: It suffices to prove that $T_i - T_{i-1} \geq R_i + R_{i-1}$, because then

$$T_i = \sum_{j=1}^{i}(T_j - T_{j-1}) \quad \geq \quad \sum_{j=1}^{i}(R_j + R_{j-1}) \quad = \quad R_i + 2\sum_{j=1}^{i-1} R_j \ ,$$

since $T_0 = 0$. To prove that $T_i - T_{i-1} \geq R_i + R_{i-1}$, consider any graph with nodes s, u, v such that $\mathsf{dist}(s, u) = R_{i-1}$, $\mathsf{dist}(s, v) = R_i$, and $\mathsf{dist}(u, v) = R_{i-1} + R_i$. (A line graph of length $R_i + R_{i-1}$ does the job.) Now, by construction, $u \in \mathsf{core}(T_{i-1}) \setminus \mathsf{core}(T_{i-1}-1)$, and $v \in \mathsf{core}(T_i) \setminus \mathsf{core}(T_i - 1)$. Hence the feasibility condition implies that (1) a broadcast from u must reach v by T_i, and (2) this broadcast can start at time T_{i-1} at the earliest. This, together with the fact that $\mathsf{dist}(u,v) = R_{i-1} + R_i$ imply the inequality. To prove the positive part of the claim, we set $T_i = T_{i-1} + R_i + R_{i-1}$ inductively. To see feasibility, note that the distance between the furthest node in the ith core and the $(i+1)$st core is at most $R_i + R_{i+1}$. Optimality in case of equality follows from induction. ▮

We have reduced the problem of finding the optimal ball core function to the problem of choosing a sequence of radii $\{R_i\}$ that maximizes the agility subject to the inequality of Lemma 3.7. Note that the agility of the algorithm is $\min_i \left\{\frac{R_{i-1}}{T_i - 1}\right\}$: at time $T_i - 1$, the core radius is still R_{i-1}.

To gain some intuition into the problem of choosing the optimal radii sequence, consider the "greedy" rule, where a node v enters the core at time t if t is the first time in which v receives the broadcasts of all nodes in $\mathsf{core}(t-1)$. This means that the radii sequence is $R_i = i$. It follows from Lemma 3.7 that the greedy rule leads to the core function $\mathsf{core}(t) = \mathsf{ball}_s(\sqrt{t})$, and hence the agility is about $\frac{1}{\sqrt{\mathsf{diam}}}$.

It is worth noting that the problem of optimizing the agility is closely related to the classical *cow path problem* [23, 15] from competitive analysis. In the cow path problem, a cow live on a straight line (path). Somewhere, in an unknown location on the path, is the cow's goal (food); the goal of an algorithm is to design a tour that will ensure that the total distance traversed by the cow is minimized with respect to the best possible, had the cow known what direction to go in the first place. In the cow's path problem, the best ratio possible is 9; in our case, the ratio is slightly better.

The following two lemmas help us choose an optimal sequence of radii. The proofs of these lemmas can also serve as new proofs for the cow path problem. The first lemma asserts a lower bound on the agility of any ball core function. The lemma actually proves the bound for infinitely many times.

Lemma 3.8 *Let $R_1, R_2, \ldots$ be any positive non-decreasing sequence such that $R_1 \geq 1$. Let $T_1, T_2, \ldots$ be a sequence such that $T_i \geq R_i + 2\sum_{j=1}^{i-1} R_j$. Then $\liminf \frac{R_i}{T_{i+1}-1} \leq \frac{1}{3+2\sqrt{2}}$.*

Proof: Fix a sequence $\{R_i\}$. Note that $T_{i+1} \geq R_{i+1} + 2\sum_{j=1}^{i} R_j \geq (1+2i)R_1 > i+1$. Thus,

$$\begin{aligned} \frac{R_i}{T_{i+1}-1} &= \frac{R_i}{T_{i+1}} + \frac{R_i}{T_{i+1}(T_{i+1}-1)} \\ &\leq \frac{R_i}{T_{i+1}} + \frac{1}{T_{i+1}-1} \qquad < \quad \frac{R_i}{T_{i+1}} + \frac{1}{i} \ , \end{aligned}$$

where the first inequality follows from the fact that $R_i \leq T_{i+1}$. Next, using the fact that for any two sequences $\{X_i\}_{i>0}$ and $\{Y_i\}_{i>0}$ we have

$$\liminf (X_i + Y_i) \leq \liminf X_i + \limsup Y_i \ ,$$

we deduce that

$$\liminf \frac{R_i}{T_{i+1}-1} \leq \liminf \frac{R_i}{T_{i+1}} + \limsup \frac{1}{i} = \liminf \frac{R_i}{T_{i+1}} \ .$$

Thus, it is enough to show that $\liminf \frac{R_i}{T_{i+1}} \leq \alpha_0$, where $\alpha_0 \stackrel{\text{def}}{=} \frac{1}{3+2\sqrt{2}}$.

Assume by contradiction that $\liminf \frac{R_i}{T_{i+1}} > \alpha_0$. Then there exists some positive N, such that for all $i > N$ we have

$$\frac{R_i}{T_{i+1}} \geq \beta \tag{1}$$

for some fixed $\beta > \alpha_0$. Let $S_i \overset{\text{def}}{=} \sum_{j=1}^{i} R_j$. Using Eq. (1) with the definitions of T_i and S_i, we conclude that

$$\beta \leq \frac{R_i}{2S_i + R_{i+1}} = \frac{S_i - S_{i-1}}{2S_i + (S_{i+1} - S_i)} = \frac{S_i - S_{i-1}}{S_i + S_{i+1}} = \frac{1 - S_{i-1}/S_i}{1 + S_{i+1}/S_i} \ . \tag{2}$$

Let us rewrite Eq. (2) in two steps as follows. First, let $y_i \overset{\text{def}}{=} S_i/S_{i-1}$. Then Eq. (2) can be written as

$$\beta \leq \frac{1 - 1/y_i}{1 + y_{i+1}} \ .$$

Next, define $z_i \overset{\text{def}}{=} 1 + y_i$ (clearly $z_i > 1$) and obtain

$$\beta \leq \frac{1 - 1/(z_i - 1)}{z_{i+1}} \ . \tag{3}$$

Now, we use the following inequality which holds for any $t > 1$:

$$1 - 1/(t-1) \leq \alpha_0 t \ . \tag{4}$$

To prove Eq. (4), note that it is equivalent (multiplying by $t-1$) to $t - 2 \leq \alpha_0 t^2 - \alpha_0 t$, i.e., Eq. (4) is equivalent to $\alpha_0 t^2 - (\alpha_0 + 1)t + 2 \geq 0$. The last inequality is easy to verify: the determinant of the quadratic solution is $(\alpha_0 + 1)^2 - 8\alpha_0$, which equals 0 by our choice of α_0.

Plugging Eq. (4) into Eq. (3), we get

$$\beta \quad \leq \quad \frac{1 - 1/(z_i - 1)}{z_{i+1}} \quad \leq \quad \frac{\alpha_0 z_i}{z_{i+1}} \ ,$$

and therefore $z_{i+1}/z_i \leq \alpha_0/\beta < 1 - \epsilon$ for some fixed $\epsilon > 0$. It follows that for large enough i, z_i becomes smaller than 1 which is a contradiction. This completes the proof of the lemma. ▮

The next lemma shows how to pick a sequence of radii that attains the agility lower bound of Lemma 3.8.

Lemma 3.9 *Let $R_i = \left\lfloor (1 + \sqrt{2})^{i+1} \right\rfloor$ for $i \geq 1$, and let $T_i = R_i + 2\sum_{j=1}^{i-1} R_j$. Then $\inf \frac{R_i}{T_{i+1} - 1} \geq \frac{1}{3 + 2\sqrt{2}}$.*

Proof: Let $\alpha_0 = \frac{1}{3+2\sqrt{2}}$. Clearly $\frac{R_i}{T_{i+1}-1} \geq \frac{R_i}{T_{i+1}}$ for all i. We show that $\frac{R_i}{T_{i+1}} \geq \alpha_0$ for all i.

We first define a sequence R'_i of real numbers, and then analyze the sequence R_i. Let $q = 1 + \sqrt{2}$. Define $R'_i = q^{i-1}$ for all $i > 0$ and $T'_i = R'_i + 2S'_{i-1} = q^{i-1} + 2(q^{i-1} - 1)/(q - 1)$. Let $S'_i = \sum_{j=1}^{i} R'_j$ as before. With these definitions we have

$$\frac{R'_i}{T'_{i+1}} = \frac{q^{i-1}}{q^i + 2\frac{q^i - 1}{q-1}} \geq \frac{q^{i-1}}{q^i + \frac{2q^i}{q-1}} = \frac{1}{q + \frac{2q}{q-1}} = \alpha_0$$

by our choice of q (this is, of course, the optimal choice for q to maximize the ratio). This completes the proof for the sequence of reals $\{R'_i\}_{i>0}$.

We now consider the integer sequence $\{R_i\}_{i>0}$ defined by $R_i = \lfloor q^{i+1} \rfloor$. By definition, $R_i = \lfloor R'_{i+2} \rfloor \geq R'_{i+2} - 1$ for all $i > 0$. Let $S_i = \sum_{j=1}^{i} R_i$. Clearly $S_i \leq S'_{i+2} - (1+q)$, since we omitted the first two elements of the sequence, and rounded the other elements down. Hence

$$T_{i+1} = R_{i+1} + 2S_i \leq R'_{i+3} + 2S'_{i+2} - 2(1+q) = T'_{i+3} - 2(1+q) \ .$$

Now, since $2(1+q)\alpha_0 > 1$ we conclude that

$$R_i \geq R'_{i+2} - 1 \geq \alpha_0 T'_{i+3} - 1 \geq \alpha_0(T'_{i+3} - 2(1+q)) \geq \alpha_0 T_{i+1}$$

where the second inequality follows from the fact that we proved the lemma for the sequence R'_i with real numbers. This completes the proof. ▮

Setting $\mathsf{core}(t) = \mathsf{ball}_s(\max\{R_i \mid t \leq T_i\})$ in Algorithm bootstrap, where the R_i and the T_i values are as given by Lemma 3.9, yields an error-confined algorithm for BCAST with agility $\frac{1}{3+2\sqrt{2}} \approx 0.172$, which is optimal for ball cores by Lemma 3.8.

4 Discussion

Error confinement, often required in centralized systems, seems to be even more useful in distributed settings. This paper represents only a first step in defining and exploring this notion in distributed systems. It leaves many problems open. Can the agility of our algorithm be improved? Our algorithm uses ball cores. Is this choice optimal? Clearly, there exist topologies and source nodes locations for which there exist better core functions. Another natural choice is that of monotonically growing cores. In some settings, certain nodes are better protected than others, and this property of nodes may change over time. Our algorithms start with a core that consists of the source only. It may be the case (as in [22]) that the initial core contains multiple nodes.

Another direction is the relaxation of the model assumptions. For example, it is interesting to investigate the case where faults hit in multiple batches. Other questions lie in correlating error confinement to other notions. For example, the notion of time adaptivity [18, 14, 17] restricts the allowed *time* span of the faulty behavior, as a function of the number of faulty nodes. Is there a trade off between such a requirement and that of error confinement, that restricts the *spatial* span of faulty behavior as a function of the number of faults?

We have shown in Theorem 3.2 that error confinement implies a slowdown in the broadcast. This leaves open other questions of overhead implied by confinement. In particular, our algorithms use much more communication resources and memory than algorithms that are not error confined. Can this overhead be reduced? The notion of error confinement presented here applies only to the external behavior. In message passing models it seems that there is no way to prevent the contamination of parts of the state that are not exposed externally. Is this possible in other models?

Finally, there is the question regarding the applicability of the results to general reactive systems. There is a trivial reduction from a general reactive task to broadcast. In this reduction, every node broadcasts its values, and every node can compute the output based on the inputs of all nodes. This, of course, is not a very efficient reduction. It would be interesting to investigate the fault confinement of other problems, as well as the agility that can be achieved for other problems. We believe that our broadcast algorithm will prove a useful primitive for solving such problems.

References

[1] Baruch Awerbuch. Complexity of Network Synchronization. J. ACM 32(4): 804-823 (1985).

[2] Y. Afek and A. Bremler. Self-stabilizing unidirectional network algorithms by power-supply. In *Proc. of the 8th ann. ACM-SIAM Symposium on Discrete Algorithms*, pages 111–120, 1997.

[3] Y. Afek and S. Dolev. Local stabilizer. In *Proceedings of the 5th Israel Symposium on Theory of Computing and Systems*, June 1997.

[4] Y. Afek, S. Kutten, and M. Yung. Memory-efficient self-stabilization on general networks. In *Proc. 4th Workshop on Distributed Algorithms*, pages 15–28, Italy, Sept. 1990. Springer-Verlag (LNCS 486). To appear in *Theoretical Comp. Sci.*

[5] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese. Time optimal self-stabilizing synchronization. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, California*, pages 652–661, May 1993. Also appeared as IBM Research Report RC-19149(83418).

[6] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico*, pages 268–277, Oct. 1991.

[7] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev. Self-stabilization by local checking and global reset. In *Proc. 8th International Workshop on Distributed Algorithms*, pages 326–339. Springer-Verlag (LNCS 857), 1994.

[8] R. Baeza-Yates, J. Culberson, and G. Rawlin Searching in the plane. Information and Computation, Vol. 106, No. 2, 1993, pp. 234-252.

[9] M. Breitling. Modeling faults of distributed, reactive systems. In *6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, (FTRTFT 2000)*, pages 58–69. Springer (LNCS 1926), 2000.

[10] Imrich Chlamtac, Shlomit S. Pinter Distributed Nodes Organization Algorithm for Channel Access in a Multihop Dynamic Radio Network. IEEE Transactions on Computers 36(6): 728-737 (1987)

[11] A. Bui, A. K. Datta, F. Petit, and V. Villain. State-optimal snap-stabilizing PIF in tree networks. In *Proceedings of the Third Workshop on Self-Stabilizing Systems (WSS 3)*, pages 78–85. IEEE Computer Society, 1999.

[12] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Comm. ACM*, 17(11):643–644, November 1974.

[13] S. Dolev and T. Herman. Superstabilizing protocols for dynamic distributed systems. In *Proc. of the Second Workshop on Self-Stabilizing Systems*, pages 3.1–3.15, May 1995.

[14] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. In *Proc. 9th Ann. ACM Symp. on Principles of Distributed Computing*, Quebec City, Canada, Aug. 1990.

[15] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *Proc. 15th Ann. ACM Symp. on Principles of Distributed Computing*, May 1996.

[16] M.-Y. Kao, J. Reif, and S. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *Proc. of the 4th ann. ACM-SIAM Symposium on Discrete Algorithms*, 1993.

[17] S. Katz and K. J. Perry. Self-stabilizing extensions for message-passing systems. *Distributed Computing*, 7(1):17–26, 1993.

[18] S. Kutten and B. Patt-Shamir. Stabilizing time-adaptive protocols. *Theoretical Computer Science*, 220(3):93–111, 1999.

[19] S. Kutten and D. Peleg. Fault-local distributed mending. In *Proc. 14th Ann. ACM Symp. on Principles of Distributed Computing*, Aug. 1995.

[20] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM*, 21(7):558–565, July 1978.

[21] P. A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice.* Springer-Verlag, Wien, second revised edition, 1990.

[22] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann, San Mateo, CA, 1995.

[23] D. Malkhi, Y. Mansour, and M. Reiter. Diffusing without false rumors: On propagating updates in a byzantine environment. *Theoretical Comput. Sci.*, 2002.

[24] C. H. Papadimitriou and M. Yanakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[25] G. Parlati and M. Yung. Non-exploratory self-stabilization for constant-space symmetry-breaking. In J. van Leeuwen, editor, *Proceedings of the 2nd Annual European Symposium on Algorithms*, pages 26–28, Sept. 1994. LNCS 855, Springer Verlag.

[26] D. J. Taylor. Practical techniques for damage confinement in software. In *Proceedings of the 1998 Computer Security Dependability and Assurance (CSDA '98).* IEEE, 1998.

[27] G. Varghese. Self-stabilization by counter flushing. *SIAM J. Comput.*, 30(2):486–510, 2000.

[28] I-Ling Yen: A Highly Safe Self-Stabilizing Mutual Exclusion Algorithm. *Information Processing Letters* 57(6): 301-305 (1996).

[29] Yechiam Yemini. "The network Book, Chapter 3, Section 3, Routing in Large Networks." Columbia University, http://www1.cs.columbia.edu/netbook03/Chapter03/Section03/03-03.pdf