Local Containment in Minimum Distance Routing Tree *

Hongwei Zhang Anish Arora †
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210 USA

Abstract

We present an algorithm for local containment and stabilization in minimum distance routing. By *local containment*, the impact of perturbations in a system is contained around the perturbed regions within distance proportional to the perturbation size; by *local stabilization*, the system stabilizes within time proportional to the perturbation size. Our algorithm achieves local containment and stabilization through two mechanisms. First, it layers system computation into three diffusing computation waves with different propagation speeds, i.e., *stabilization wave* with the lowest speed, *containment wave* with intermediate speed, and *super-containment wave* with the highest speed. The containment wave contains potential faults in stabilization wave, the super-containment wave contains potential faults in containment wave, and there is no potential fault in super-containment wave. Second, it avoids forming cycles during stabilization. It guarantees loop freedom during stabilization if there exists no loop at the initial state, and it removes all the loops from a system within one step of computation if there exists any at the initial state. To the best of our knowledge, this is the first distributed algorithm that achieves local containment and stabilization in minimum distance routing.

Keywords: minimum distance routing, distance-vector, local containment, local stabilization, availability, stability, scalability

*This work was partially sponsored by DARPA contract OSU-RF #F33615-01-C-1901, NSF grant NSF-CCR-9972368, an Ameritech Faculty Fellowship, and two grants from Microsoft Research.

†Email: {zhangho, anish}@cis.ohio-state.edu; Tel: +1-614-292-1836; Fax: +1-614-292-2911; Web: http://www.cis.ohio-state.edu/{~zhangho, ~anish}.

1 Introduction

Simple faults, such as node crash and message loss, can drive a distributed protocol into arbitrary states [11], therefore self-stabilization [6] is a desired property of any distributed system where faults can occur. By self-stabilization, a distributed system is guaranteed to eventually converge to a legitimate state, starting at an arbitrary state. As an important aspect of fault-tolerance, self-stabilization has been a focus of research in the community of distributed computing for decades, and self-stabilizing algorithms for a rich class of problems (such as routing and leader election) have been proposed.

However, the impact of a fault-perturbed region can reach far beyond the region itself in a self-stabilizing distributed system. That is, faults can propagate to and affect nodes far away from where the faults have occurred before a system eventually stabilizes to a legitimate state. Fault propagation during stabilization degenerates the availability as well as stability of a system because a large part of or even the whole system can be affected and led into an illegitimate state due to some faults that only happened at a small part of the system (for example, faults at a single node). Moreover, fault propagation degenerates the scalability of a system, because the time taken for the system to stabilize can be a function of the system size instead of the size of nodes where faults initially occur. Therefore, in order to increase the availability, stability, and scalability of distributed systems, especially for large-scale distributed systems [17][14], it is desirable that faults are contained locally around the region where those faults occurred, and nodes far away from those faults are unaffected (this property is called local containment in this paper). It is also desirable that the time taken for a system to stabilize is only a function of the size of the fault-perturbed regions instead of the size of the system (this property is called local stabilization in this paper).

One problem where local containment and stabilization is critical is minimum distance routing in networks. minimum distance routing is a critical issue in distributed systems since it affects the quality of communication between different nodes in a distributed system. Routing algorithms proposed so far can be classified as distance-vector based or link-state based. In link-state based algorithms, each node maintains the topological information about a whole network, and local containment is impossible in these algorithms. In distance-vector based algorithms, instead of maintaining the topological information about a whole network, each node only maintains the distance of and the next-hop on its shortest path to each destination in the network. Therefore, local containment and stabilization are possible in distance-vector based routing algorithms.

Distance-vector based routing protocols for Internet, such as RIP and BGP, have been studied and implemented for decades [10]. Distance-vector based routing protocols for mobile ad hoc networks, such as DSDV and AODV, have been proposed too [15]. However, researchers have only concentrated on how to avoid routing loops and count-to-infinity problem when designing these algorithms, and neither local containment nor local stabilization is guaranteed in these algorithms. When a node fails, all the nodes whose shortest path include this failed node can be affected even though they don't have to. And this can result in severe instability in Internet routing as shown in [5]. Also, when these algorithms were designed, only such faults as node-fail-stop and new-node-join were considered, and state corruption was not considered. However, mis-configuration and faulty software (a special kind of state corruption) are the major causes for routing instability in Internet [4]. Therefore, local containment and stabilization are desired in order to increase the availability, stability, and scalability of distance-vector based routing in large-scale networks

such as Internet and wireless sensor networks [17].

Contributions of the paper In this paper, we design an algorithm for local containment and stabilization in minimum distance routing. By local containment and stabilization, the time taken for a system to stabilize and the range of contamination during stabilization is $O(p)$, when the system starts at an arbitrary state where the size of perturbation is p. Our algorithm achieves local containment and stabilization by avoiding forming cycles during stabilization, and by layering system computation into three waves with different propagation speeds, that is, stabilization wave with the lowest speed, containment wave with intermediate speed, and super-containment wave with the highest speed. The stabilization wave eventually corrects any corrupted state at perturbed nodes in a system, the containment wave contains potential faults in the stabilization wave, and the super-containment wave contains potential faults in the containment wave. There exists no fault in the super-containment wave because it only uses variables defined for the stabilization wave and containment wave, and no variable is introduced for the super-containment wave. To the best of our knowledge, our algorithm for minimum distance routing is the first distributed routing algorithm that guarantees local containment and stabilization. We also believe that these algorithmic paradigms introduced to achieve local containment and stabilization are applicable to a richer class of distributed computing problems with the flavor of diffusing computation.

Moreover, our algorithm guarantees loop freedom during stabilization if there exists no loop at the initial state, and it removes all the loops from a system within one step of computation if there is any at the initial state. Our algorithm is also scalable in space in the sense that each node only maintains information about its 1-hop neighbors.

We also discuss the impact of network topology on local containment and stabilization. We observe that denser edges can make the perturbation size as well as the range of contamination during stabilization smaller upon perturbations in a system.

In order to precisely characterize the local containment and stabilization properties of algorithms, we define the concepts of perturbation size, range of contamination, $\mathcal{F}$-local containing, and $\mathcal{F}$-local stabilizing. Perturbation size describes the severity of perturbations in a system at certain state, and it is the number of nodes in the largest contiguously perturbed region in a system. Range of contamination describes the impact of perturbations in a system, that is, the maximum minimum distance (in terms of the number of hops) from a node that is not perturbed initially but affected during stabilization, to its closest node that is perturbed initially. A system is $\mathcal{F}$-local containing if the range of contamination is a function $\mathcal{F}$ of the perturbation size. And a system is $\mathcal{F}$-local stabilizing if, starting at an arbitrarily perturbed state, the time taken for the system to stabilize to a legitimate state is a function $\mathcal{F}$ of the perturbation size at that state. These concepts are independent of the minimum distance routing problem, and are applicable to any distributed computing problems.

The rest of the paper is organized as follows. In Section 2, the system model is presented, the concepts related to local containment and stabilization, and the problem of local containment and stabilization in minimum distance routing are formally defined. In Section 3, the algorithm for local containment and stabilization in minimum distance routing is presented. In Section 4, we analyze the properties of our algorithm. In Section 5, we discuss the impact of network topology on local containment and stabilization. Related work is discussed in Section 6. And conclusion is drawn in Section 7.

2 System model, Definitions, and Problem statement

In this section, we describe the system model adopted in our work, define the concepts related to local containment and stabilization, and propose the problem of local containment and stabilization in minimum distance routing.

2.1 System Model

System Nodes A system is represented as a connected undirected graph $G = (V, E)$, where V and E are the set of nodes (e.g. processes) and the set of edges in the system respectively. Each node in the system has a unique ID, and a node is referred to by its ID. If nodes i and j can communicate with each other directly, then edge (i, j) is in E. The weight of each edge is 1.

> **Communication Model** Two neighboring nodes in the system communicate with each other through a shared memory. A node can read and write variables of itself, and it can read variables of its neighbors. Also, a node can read variables of all its neighboring nodes simultaneously [1]. But a node cannot write variables of its neighbors.
>
> **Partial Synchrony** Each node has a clock, and the difference between clock rates at different nodes is bounded from above by a factor of B_c. That is, if r_1 and r_2 are the clock rates at two different nodes, then $\frac{1}{B_c} \leq \frac{r_1}{r_2} \leq B_c$.

Fault Model The state of nodes can be corrupted, nodes can fail-stop, and new nodes can join a system. However, we assume that the program at a node cannot be corrupted in spite of all these possible faults in the system.

Programming Notation We write programs using a guarded command notation: each process consists of a finite set of variables and a finite set of actions. Each action consists of two parts: a guard and a statement. For convenience, we associate each action with a unique name. Thus, each action has the following form:

$$\langle name \rangle :: \langle guard \rangle \longrightarrow \langle statement \rangle$$

The guard is a boolean expression over the variables of that and possibly other processes, and the statement updates zero or more variables of that process. An action is executed only if it is enabled, i.e., if its guard evaluates to true. To execute the action, its statement is executed atomically.

A variant of the above form is:

$$\langle name \rangle :: \langle guard \rangle \xrightarrow{d} \langle statement \rangle$$

An action of this form is executed at time t if its guard evaluates to true at t and t is at least d amount of time after the moment when the guard first evaluates to true.

Computation Model Every computation of a system is regarded as a sequence of *Step*s. In each step, every node in the system executes one program action if there is at least one being enabled at the end of the previous step at that node. And nodes within 1-hop neighborhood take turns to execute their actions if there is any being enabled [1].

[1] This can be achieved through the algorithm for dining philosophers [14] that also contains fault locally.

2.2 Definitions

In this subsection, we define concepts related to local containment and local stabilization. These definitions are generic for any distributed computing problems, and are not confined to the problem of minimum distance routing.

In a distributed system, the variables or states each node maintains depend on the algorithm used in the system. Among all the variables maintained at a node, some of them are inherent with the problem itself and independent of algorithms being used, and the others are dependent on algorithms being used. For example, in the problem of minimum distance routing, each node has to maintain the next-hop on its shortest path to each destination node in order to be able to send a message to the destination, and the variable used to record this next-hop is inherent with the problem of minimum distance routing. We call those variables that are inherent with the problem in a distributed system *central variables*. Moreover, when faults occur in a distributed system, a node where no fault has occurred may still have to change the values of its central variables in order for the system to stabilize to a legitimate state no matter what algorithm is used, due to the dependency between different nodes in the system. For example, in the problem of minimum distance routing, if a node i fail-stops, then every node whose only shortest path to a destination goes through node i before i fail-stops will have to change their state, that is, change its next-hop on its shortest path to the destination.

In a distributed system, a node i is said to be *perturbed* at a system state q if either of the following conditions holds:

- Some transient fault (such as state corruption) occurred at node i when the system was at a state q' ($q' \neq q$), the system has not reached any legitimate state before it reaches state q starting at q', and node i needs to change its state in order for the system to converge to a legitimate state starting at q.
- No transient fault occurred at node i before the system reaches state q starting at a legitimate state q'', but node i needs to change the value(s) of its central variable(s), due to faults at some other nodes before the system reaches state q, in order for the system to converge to a legitimate state starting at q.

A node is said to be *healthy* at a system state q if it is not perturbed at state q.

A set of nodes S are said to be *contiguous* if the subgraph of G on S is connected, i.e., the graph $G'(V', E')$, where $V' = S$ and $E' = \{(v_1, v_2) : v_1 \in S \wedge v_2 \in S \wedge (v_1, v_2) \in E\}$, is connected; otherwise, they are said to be *non-contiguous*. A set of perturbed nodes that are contiguous is called a *perturbed region*.

Then the Perturbation Size for a system can be defined as follows.

Definition 1 (Perturbation Size) *Given an arbitrary state q, the perturbation size for a system $G(V, E)$ at state q, denoted as $P(G, q)$, is defined for a single node, for a perturbation region, and for the system G as follows:*

- *Perturbation size for a single node i ($i \in V$):*

$$P(i, q) = \begin{cases} 1 & \textit{If } i \textit{ is perturbed at state } q \\ 0 & \textit{If } i \textit{ is healthy at state } q \end{cases}$$

- *Perturbation size for a perturbed region S ($S \subseteq V$):*

$$P(S,q) \ = \ | \, S \, |$$

- *Perturbation size for the system G:*

$$P(G,q) = max\{P(S,q) : S \subseteq V \ \wedge \ S \textit{ is a perturbed region at state } q\}$$

Intuitively, the perturbation size denotes the minimum amount of work needed to correct a perturbation that occurs in a system, and it only depends on the problem being considered.

After defining the perturbation size that is problem-inherent, let us define the concepts related to properties of algorithms, that is, range of contamination, $\mathcal{F}$-local containing, and $\mathcal{F}$-local stabilizing.

Definition 2 (Range of Contamination) *When a system*[2] $G(V,E)$ *starts at an arbitrary state* q*, the range of contamination during stabilization, denoted as* $R_c(G,q)$*, is the maximum distance from the set of nodes that are healthy at* q *but affected during stabilization to the set of perturbed nodes at* q*. That is,*

$$\begin{aligned} R_c(G,q) \quad = \quad & max\{dist(i,S_p) : i \in S_c\}, \textit{where} \\ & S_p = \{j : j \in V \wedge j \textit{ is perturbed at state } q\} \\ & S_c = \{k : k \in V \wedge k \textit{ is healthy at state } q \wedge \textit{ at least one action is executed at } k \textit{ after state} q\} \\ & dist(i,V') = min\{dist(i,j,G) : j \in V'\} \\ & dist(i,j,G) = \textit{the minimum distance (i.e., the number of hops on the shortest path) between} \\ & \qquad\qquad i \textit{ and } j \textit{ in } G \end{aligned}$$

Definition 3 ($\mathcal{F}$-Local Containing) *A system $G(V,E)$ is $\mathcal{F}$-Local Containing if and only if*

> *Starting at an arbitrary state q, the system stabilizes to a legitimate state, and the range of contamination during stabilization is a function $\mathcal{F}$ of the perturbation size for the system at state q, i.e., $R_c(G,q) = \mathcal{F}(P(G,q))$.*

Definition 4 ($\mathcal{F}$-Local Stabilizing) *A system $G(V,E)$ is $\mathcal{F}$-Local Stabilizing if and only if*

> *Starting at an arbitrary state q, the system stabilizes to a legitimate state within time that is a function $\mathcal{F}$ of the perturbation size for the system at state q, i.e., $\mathcal{F}(P(G,q))$.*

If a system is $\mathcal{F}$-local containing or $\mathcal{F}$-local stabilizing, we say that local containment or local stabilization is guaranteed (or achieved) in the system.

Now, we are ready to define the problem of local containment and stabilization in minimum distance routing.

[2] *The algorithm that is used in a system is considered as a part of the system.*

2.3 Problem statement

Informally, the problems of minimum distance routing is to construct and maintain a spanning tree T of a system $G(V, E)$, such that a destination node r is the root of T, and the path from any other node $i (i \neq r)$ to r in T is a shortest path between i and r in G. Moreover, local containment and local stabilization are required, that is, both the range of contamination and the time taken for the system to stabilize from an arbitrary state should be a function of the perturbation size at the initial state.

Formally, the problem is to design an algorithm that, given a connected graph $G(V, E)$ and a destination node $r (r \in V)$, constructs and maintains a spanning tree T (called *shortest path tree*) of G that meets the following requirements:

a) Node r is the root of T;

b) $(\forall i : i \in V \Rightarrow dist(r, i, T) = dist(r, i, G))$, where $dist(r, i, G')$ is the minimum distance (i.e., the number of hops on the shortest path) between nodes r and i in a graph G';

c) Starting at an arbitrary state q_0, the system stabilizes to a legitimate state q_f, and the time taken for the system to stabilize to q_f as well as the range of contamination during stabilization is $\mathcal{F}(p)$, where $\mathcal{F}$ is a function and $p = P(G, q_0)$.

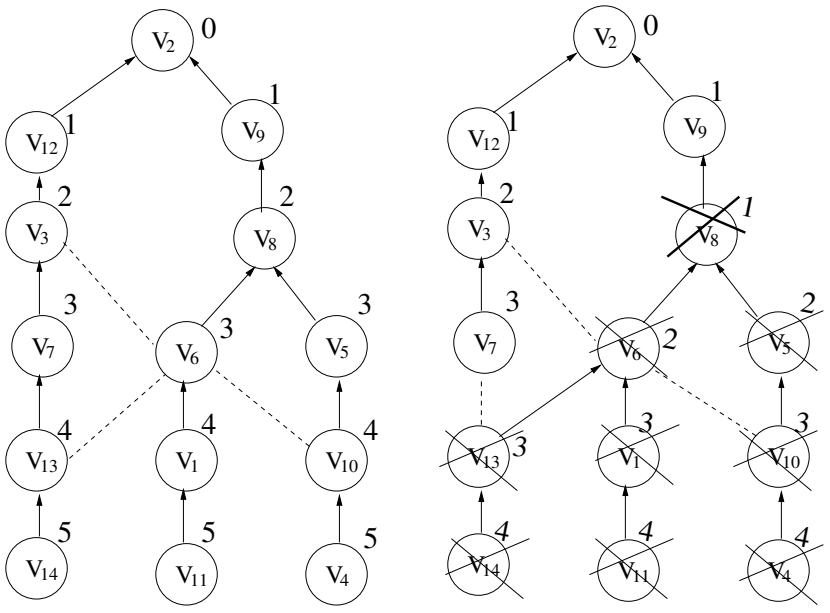
3 Algorithm

In this section, we explain the problems with existing distance-vector based routing algorithms, present the algorithmic concepts behind our algorithm for local containment and stabilization in minimum distance routing, and discuss in detail our algorithm design.

3.1 Problems with existing distance-vector based routing algorithms

All existing distance-vector based routing algorithms are based on the distributed Bellman-Ford algorithm [13]. In those algorithms [10][7], each node i maintains the distance (denoted as $d.i$) of and the next-hop (denoted as $p.i$) on its shortest path to each destination. For a destination r, if node j is a neighbor of node i and $d.j = min\{d.k : k \text{ is a neighbor of } i\}$, node i will choose j as the next hop on its shortest path to r (i.e., set $p.i$ to j) and set $d.i$ to $(d.j + w_{ij})$, where w_{ij} is the weight of edge (i, j). However, in these algorithms, faults cannot be contained around where they have occurred, and they can propagate unbounded.

One example is shown in Figure 1 where node v_2 is the destination. If existing Bellman-Ford based routing algorithms are used, Figure 1(a) represents a legitimate system state. Now, if the state of node v_8 is corrupted such that $d.v_8 = 1$. Then nodes v_6 and v_5 will first detect the change of $d.v_8$, but $d.v_8$ is still the smallest "distance value" among their neighbors. So both v_6 and v_5 will change their own state correspondingly such that $d.v_6 = d.v_8 + 1 = 2$ and $d.v_5 = d.v_8 + 1 = 2$. And the same thing happens to nodes v_1, v_{10}, v_{11}, v_4, v_{13}, and v_{14}. The perturbed system state after the fault propagation from v_8 is shown in Figure 1(b). Even though the system can stabilize to a legitimate



(a) A legitimate state

(b) A perturbed state after v_8 is corrupted

Figure 1: Example of fault propagation in existing distance-vector based routing algorithms. In the Figure, each circle represents a node in the system, the string in the circle represents the ID of this node, and the number besides the circle represents the distance from the node to the destination (node 2 in this example). A directed edge $< i, j >$ means that $p.i = j$, and an undirected dotted edge (i, j) means that node i and j are neighbors in the system but neither $p.i = j$ nor $p.j = i$.

state later, nodes far away from v_8, such as v_{14} and v_{11}, have been affected by the state corruption at v_8, especially node v_{13} has changed its route to destination v_2 because of the fault propagation.

In a large-scale system, there can be multiple faults happening at different nodes in the system, and the fault at any single node can propagate across and affect the whole system. This kind of unbounded fault propagation can result in severe routing instability [4] and reduce the availability as well as scalability of a system. Therefore, algorithms that can contain faults locally around where they have occurred and let a system stabilize in time that is only a function of the perturbation size are strongly desirable in these fault-prone systems.

3.2 Algorithmic concepts

In the example shown in Figure 1, the reason that the state corruption at node v_8 can propagate far away until reaching nodes which are the farthest from the destination node v_2 is as follows:

- First, the "distance value" of v_8 (i.e., $d.v_8$) is corrupted to be smaller than it should be in a legitimate state;
- Second, before node v_8 corrects its corrupted state, node v_6 and v_5 detects the state change at v_8. Because neither

v_6 nor v_5 knows that the new state of v_8 is a corrupted one, v_6 as well as v_5 only finds that $d.v_8$ decreases[3] and it is still the smallest distance value among all its neighbors. Therefore, both v_6 and v_5 update their state according to the corrupted state of node v_8, and the fault (state corruption in this case) at v_8 propagates to its neighbors v_6 and v_5. Then, the same thing that has happened to node v_6 and v_5 happens to nodes that are neighbors of v_6 and v_5, and so on.

- Third, after detecting that its state has been corrupted, node v_8 corrects its state (i.e., sets $d.v_8$ to 2). Then, its neighbors v_6 and v_5 corrects their corrupted state, and so on. However, this "correction" action is unable to catch up with the "fault propagation" action that propagates the initial corruption at v_8. Therefore, the initial corruption at v_8 is propagated far away.

According to the analysis above, the reason that faults propagate in a self-stabilizing system is that the "correction" action always lags behind the "fault propagation" action. Therefore, in order to contain faults locally, we should design an algorithm such that the node, which is the source of a fault propagation, will detect the existence of the fault propagation, and initiate the "correction" action that can catch up with and stop the "fault propagation" action before faults propagate far away.

Before looking at our solution to this, let us first look at the property of a diffusing computation. Conceptually, a diffusing computation is a process of propagating certain actions from an originator (or a set of originators) across a system. That is, a diffusing computation can be regarded as a wave that propagates from the originator(s) outward to the rest of a system, and actions belonging to the diffusing computation are executed at a node whenever the wave first reaches that node. In those Bellman-Ford based routing algorithms, the diffusing computation is a wave that propagates outward from the destination node r, and sets the distance of as well as the next-hop on the shortest path from a node i to r when the wave first reaches i.

In order to achieve local containment and local stabilization in minimum distance routing, we divide the diffusing computation involved in minimum distance routing into three diffusing waves: *stabilization wave*, *containment wave*, and *super-containment wave*. Each wave has different propagation speed, with stabilization wave of the lowest speed, containment wave of intermediate speed, and super-containment wave of the highest speed.

A stabilization wave is a diffusing computation that implements the basic distributed Bellman-Ford algorithm with some changes in order to cooperate with containment wave. A stabilization wave can propagate the "correction" action that makes a system converge to a legitimate state, but it can also propagate faults far away from where the faults initially happened, as we have seen in Figure 1. Therefore, in order to prevent a stabilization wave propagating faults unboundedly, a containment wave is introduced.

A containment wave is initiated from a node that is a "source of corruption propagation", that is, the destination node whose distance value is not 0, or a node whose distance value is no greater than that of any of its neighbors but it is not the destination node (for example, node v_8 is a source of corruption propagation in Figure 1(b)). A containment wave propagates faster than a stabilization wave in order for the containment wave to be able to catch up with and stop a stabilization wave. However, it is possible that a containment wave is "mistakenly" initiated. For example, in

[3]This means that node v_8 has chosen a shorter path if no state corruption had happened.

Figure 1(a), if the state of v_8 is corrupted such that $d.v_8 = 4$, node v_5 will become a source of corruption propagation and a containment wave will be initiated at v_5. Therefore, in order to prevent the "mistakenly" initiated containment wave propagating unboundedly, a super-containment wave is introduced.

A node that has mistakenly initiated a containment wave will detect that it should not have initiated the containment wave after time proportional to the perturbation size at most. Then it will initiate a super-containment wave that will catch up with and stop the mistakenly initiated containment wave, because the super-containment wave propagates faster than the containment wave. Moreover, there exists no potential fault in a super-containment wave because it only uses variables defined for the stabilization wave and containment wave, and no extra variable is introduced for the super-containment wave.

Besides containing faults locally, we also want a system to stabilize fast, that is, the time taken for a system to stabilize should be a function of the perturbation size instead of something else (such as system size). However, the basic distributed Bellman-Ford algorithm has the problem of bouncing effect and count-to-infinity [10] that can delay the stabilization of a system. Therefore, in order to avoid these two problems, our algorithm avoids forming cycles during stabilization, which guarantees that the stabilization time is only a function of the perturbation size in the worst case.

3.3 Algorithm design

The algorithm for local containment and stabilization in minimum distance routing is shown in Figure 2.

There are four constants in the algorithm, that is, r, d_s, d_c, and d_{sc}. r is the ID of the destination node in a system to which all the other nodes in the system need to find the shortest path. d_s, d_c, and d_{sc} are used to control the propagation speed of stabilization wave, containment wave, and super-containment wave respectively. In order to guarantee local containment and stabilization, d_s, d_c, and d_{sc} should be such that $d_s > d_c > d_{sc} \geq 0$, $(d_s - d_c) > (K \cdot d_0 - d'_0)$, and $(d_c - d_{sc}) > (K \cdot d_0 - d'_0)$, where K is the maximum degree of nodes in the system, d_0 and d'_0 are the upper bound and lower bound for the execution time of a single action in the system respectively [4].

Each node i in the system maintains three variables: $d.i$, $p.i$, and $ghost.i$. $d.i$ records the shortest distance from i to r. $p.i$ records the next-hop on the shortest path from i to r, that is, the parent of i in the shortest path tree rooted at r. $ghost.i$ is $true$ if node i is being involved in a containment wave. For convenience of presentation, we let $N.i$ denote the set of neighboring nodes of node i.

As we mentioned in Section 3.2, the diffusing computation for local containment and stabilization in minimum distance routing consists of three diffusing waves: stabilization wave, containment wave, and super-containment wave. Among the six actions in our algorithm, actions S_1 and S_2 belong to stabilization wave, C_1 and C_2 belong to containment wave, and SC_1 and SC_2 belong to super-containment wave.

Stabilization wave guarantees that a system eventually stabilizes to a legitimate state, and it implements the basic Bellman-Ford algorithm with some changes to cooperate with containment wave. The intuition behind actions S_1 and S_2 for stabilization wave is as follows.

- **Action** S_1: if a node i is a minimal point of the system (i.e., $MP.i = true$) but $p.i \neq i$, set $p.i$ to i.

[4] The reason for this is shown in the proof for Lemma 1.

Program $LCS_Minimum_Distance_Routing$
Constant $r : node_id$
$d_s,\ d_c,\ d_{sc} : real$
Parameter $j : node_id$
Var $d.j : integer$
$p.j : node_id$
$ghost.j : boolean$
$k : node_id$

Node i
Begin

$\langle S_1 \rangle$ :: $MP.i \wedge p.i \neq i \longrightarrow p.i := i$
[]
$\langle S_2 \rangle$ :: $SW.i.j \wedge \neg ghost.j \xrightarrow{d_s} d.i, p.i := d.j + 1, j;$
$ghost.i := false$
[]
* * * * *
$\langle C_1 \rangle$ :: $\neg ghost.i \wedge (SP.i \vee CW.i) \xrightarrow{d_c}$ $\underline{if}\ SP.i\ \underline{then}\ p.i := i;$
$ghost.i := true$
[]
$\langle C_2 \rangle$ :: $ghost.i \wedge \neg(\exists k : k \in N.i \wedge p.k = i \wedge d.k = d.i + 1) \longrightarrow$ $\underline{if}\ i = r\ \underline{then}\ d.i := 0\ \underline{else}\ d.i := \infty;$
$p.i, ghost.i := i, false$
[]
* * * * *
$\langle SC_1 \rangle$ :: $ghost.(p.i) \wedge PS.i.j \longrightarrow d.i, p.i, ghost.i := d.j + 1, j, false$
[]
$\langle SC_2 \rangle$ :: $ghost.i \wedge SCW.i \xrightarrow{d_{sc}} ghost.i := false$

End

Figure 2: Algorithm for Local Containment and Stabilization in Minimum Distance Routing

$MP.i$ is defined as

$$(i = r \wedge d.i = 0) \vee (i \neq r \wedge ghost.i \wedge SP.i), \text{where } SP.i \text{ is defined in action } C_1$$

That is, a node i is a minimal point if it is the destination node r and $d.i = 0$, or if it is not the destination node r, but it is being involved in a containment wave and it is the initiator of the containment wave [5].

- **Action** S_2: if node i should propagate a stabilization wave from node j (i.e., $SW.i.j = true$), node j is not being involved in a containment wave, and this condition still holds d_s amount of time after the moment when it first became true, then node i sets j as its parent, and sets $d.i$ to $d.j + 1$, and $ghost.i$ to $false$.

[5] This action is required to prove Theorem 2, 4, and 5.

$SW.i.j$ is defined as

$$i \neq r \wedge j \in N.i \ \wedge \ d.j < d.i \ \wedge (\forall k : k \in N.i \Rightarrow d.j \leq d.k) \wedge$$
$$((j \neq p.i \wedge (p.i \in N.i \Rightarrow d.j < d.(p.i))) \vee (j = p.i \wedge d.i \neq d.j + 1))$$

That is, node i should propagate a stabilization wave from node j if node i is not the destination node r, node j is the neighbor of i whose distance value (i.e., $d.j$) is the smallest among that of all the neighbors of i, and either j is not the current parent of i (i.e., $p.i$) and the distance value of j is less than that of $p.i$ unless $p.i$ is not a neighbor of i, or j is the current parent of i but the distance value of j has changed.

Moreover, node i should not set a node j as its parent if node j is being involved in a containment wave, since the state of any node being involved in a containment wave is corrupted.

Because a stabilization wave can "mistakenly" propagate faults far away from where they have occurred, we need to introduce a delay d_s between the execution of action S_2 and the moment when it first became enabled, in order to control the propagation speed of the stabilization wave and for the containment wave to catch up with and stop a stabilization wave that does propagate faults.

Containment wave prevents stabilization wave propagating faults far away from where they have occurred. And it is achieved by actions C_1 and C_2. The intuition behind these two actions is as follows.

- **Action** C_1: if node i is not being involved in a containment wave (i.e., $ghost.i = false$), and it is either a source of corruption propagation (i.e., $SP.i = true$) or it should propagate a containment wave from its parent (i.e., $CW.i = true$), then node i sets $ghost.i$ to $true$ in order to initiate or propagate a containment wave. Moreover, if node i is a source of corruption propagation, it sets $p.i$ to i.

  $SP.i$ is defined as

  $$(i = r \wedge d.i \neq 0) \vee (i \neq r \wedge d.i \neq \infty \wedge d.i \neq d.(p.i) + 1 \wedge (\forall j : j \in N.i \wedge \neg ghost.j \Rightarrow d.j \geq d.i))$$

  That is, a node i is a source of corruption propagation if it is the destination node r and $d.i \neq 0$, or if it is not the destination node r, its distance value is neither ∞ nor that of its current parent plus one, but is the smallest among nodes within its 1-hop neighborhood (including itself) that are not being involved in any containment wave.

  $CW.i$ is defined as

  $$p.i \in N.i \wedge ghost.(p.i) \wedge d.i = d.(p.i) + 1 \ \wedge \neg(\exists j : PS.i.j), \text{where } PS.i.j \text{ is defined in action } SC_1$$

  That is, a node i should propagate a containment wave from its parent $p.i$ if node $p.i$ is a neighbor of i, node $p.i$ is being involved in a containment wave, $d.i$ is equal to $d.(p.i) + 1$, and there is no parent substitute (defined in action SC_1) for node i.

  Because a containment wave can be "mistakenly" initiated, we need to introduce a delay d_c between the execution of action C_1 and the moment when it first became enabled, in order to control the propagation speed of

the containment wave and for the super-containment wave to catch up with and stop a containment wave that is "mistakenly" initiated.

Moreover, in order to enable super-containment wave that traces a mistakenly initiated containment wave by the parent-child relationship among neighboring nodes, containment wave needs to be a *round* procedure. That is, it consists of a phase of propagating outward by the execution of action C_1 and a phase of shrinking back by the execution of action C_2, and when action C_1 is executed at a node i that is not a source of corruption propagation, the pointer to its current parent (i.e., $p.i$) should be left unmodified.

- **Action** C_2: if node i is being involved in a containment wave, but there is no child node k of it that is perturbed due to the state corruption at i (i.e., $p.k = i$ and $d.k = d.i + 1$), then node i sets $d.i$ to 0 if it is the destination node r or to ∞ otherwise, and it sets $p.i$ to i, and $ghost.i$ to $false$.

  Action C_2 guarantees that a containment wave will shrink back to its initiator after the containment wave has caught up with and stopped the stabilization wave that propagates the faults which initially occurred to the initiator of the containment wave.

Super-containment wave prevents a "mistakenly" initiated containment wave propagating unboundedly, that is, it contains and corrects faults in a containment wave locally with respect to where these faults have occurred. Super-containment wave is implemented by actions SC_1 and SC_2, and the intuition behind these actions is as follows.

- **Action** SC_1: if the current parent of node i is being involved in a containment wave (i.e., $ghost.(p.i) = true$), and there exists a parent substitute j for i (i.e., $PS.i.j = true$), then node i selects j as its new parent, and set $d.i$, $p.i$, and $ghost.i$ to $d.j + 1$, j, and $false$ respectively.

  $PS.i.j$ is defined as

$$j \in N.i \wedge \neg ghost.j \wedge d.j < d.i \wedge (\forall k : k \in N.i \wedge \neg ghost.k \Rightarrow d.j \leq d.k)$$

  That is, node j is a parent substitute of node i if j is a neighbor of i, j is not involved in any containment wave, the distance value of j is less than that of i, and the distance value of j is the smallest among that of all the neighbors of i that are not involved in any containment wave.

  Action SC_1 guarantees that a containment wave will stop at a node that has a parent substitute for its current parent node.

- **Action** SC_2: if node i is currently involved in a containment wave (i.e., $ghost.i = true$), but it should initiate or propagate a super-containment wave from its parent (i.e., $SCW.i = true$), then node i sets $ghost.i$ to $false$.

  $SCW.i$ is defined as

$$(i = r \wedge d.i = 0) \vee (i \neq r \wedge (\neg SP.i \vee \neg ghost.(p.i)))$$

  That is, node i should initiate or propagate a super-containment wave if it is the destination node r and $d.i = 0$, or if it is not the destination node r, and neither is it a source of corruption propagation nor is its parent involved in any containment wave.

Also, in order to control the propagation speed of super-containment wave, we need to introduce a delay d_{sc} between the execution of action SC_2 and the moment when it first became enabled. But d_{sc} can be 0, since there is no potential fault in any super-containment wave, and we do not need another diffusing wave in case faults occur in a super-containment wave.

4 Algorithm analysis

In this section, we present the fixpoint of our algorithm, prove the properties of self-stabilization, local containment, and local stabilization of our algorithm for the problem of minimum distance routing. We also prove the properties of loop freedom during stabilization and fast loop removal of our algorithm, which are critical for routing algorithms in terms of network resource utilization as well as quality of communication.

4.1 Properties of local containment and stabilization

Let $F \equiv (\forall i : i \in V \Rightarrow \neg ghost.i \wedge LH.i)$, where

$$\begin{aligned} LH.i \quad \equiv \quad & (i = r \Rightarrow d.i = 0 \wedge p.i = i) \wedge \\ & (i \neq r \Rightarrow d.i = d.(p.i) + 1 \wedge p.i \in N.i \wedge (\forall k : k \in N.i \Rightarrow d.(p.i) \leq d.k)) \end{aligned}$$

Then, any state in F is a legitimate system state where the shortest path tree rooted at the destination node r is formed by the variable $p.i$ for every node i in the system, and every node i has learned the distance of and the next-hop on its shortest path to node r.

Let us first prove that F is a fixpoint (or stable state) of our algorithm, and the algorithm self-stabilizes to a state in F, upon starting at an arbitrary system state.

Theorem 1 (Fixpoint of the algorithm) *F is a fixpoint of the algorithm, i.e., any state in F is a stable state of the system.*

Proof: In order to prove that F is a fixpoint of the algorithm, we need to prove that no action is enabled at any state in F. Because F implies $(\forall i : i \in V \Rightarrow LH.i)$, $(\forall i : i \in V \Rightarrow LH.i)$ holds at every state in F. Thus actions S_1, S_2 and C_1 are not enabled at any state in F. Similarly, $(\forall i : i \in V \Rightarrow \neg ghost.i)$ holds at every state in F, which means that actions C_2, SC_1 and SC_2 are not enabled at any state in F.

Therefore, no action is enabled at any state in F, and F is a fixpoint of the algorithm.

□

Theorem 2 (Self-stabilization of the algorithm to F) *Starting at an arbitrary state, the system is guaranteed to reach a state in F within $O(N)$ amount of time, where N is the number of nodes in the system.*

Proof: Let D and K be the diameter and the maximal degree of nodes in the system $G(V, E)$ respectively. Also, we assume that the time taken to execute any action is bounded from above by d_0.

In order to prove the stabilization of F, we employ the "convergence stair" method as has been used in [2]. That is, we exhibit a finite sequence of state predicates $F.0, F.1, \ldots, F.D$ such that

(i) Starting at an arbitrary state, the system is guaranteed to reach a state in $F.0$.

(ii) For each l such that $0 \leq l \leq D$:

$F.l$ is closed under system execution; that is, once $F.l$ holds in an arbitrary system computation, it continues to hold subsequently.

(iii) For each l such that $0 \leq l < D$:

Upon starting at an arbitrary state in $F.l$, the system is guaranteed to reach a state in $F.(l+1)$ within a constant amount of time.

(iv) $F.D \equiv F$

For our algorithm, $F.0, F.1, \ldots, F.D$ are defined as follows,

- $F.0 \equiv (\forall i : i \in V \Rightarrow \neg ghost.i \wedge \neg SP.i \wedge d.i \geq 0)$
- For $0 \leq l < D$:

$$F.(l+1) \equiv F.l \wedge (\forall i : i \in V \wedge dist(i, r, G) = l \Rightarrow LH.i) \wedge (\forall i : i \in V \wedge dist(i, r, G) > l \Rightarrow d.i > l),$$

where $dist(i, r, G)$ = the minimum distance between i and r in G

We prove the stabilization of F by proving the individual elements of the "convergence stair" method mentioned above as follows.

(I) Starting at an arbitrary state q_0, the system is guaranteed to reach a state in $F.0$ in $O(N)$ time.

We prove this in three steps:

(a) Starting at state q_0, the system is guaranteed to reach a state in $F.0'$ in $O(N)$ amount of time, where $F.0' \equiv (\forall i : i \in V \Rightarrow \neg ghost.i \wedge \neg SP.i)$.

Since every computation in a system is regarded as a sequence of steps. We number the sequence of steps as $s_0, s_1, s_2, \ldots$ and so on. s_0 is an *empty step* where no action has been executed, and it is the starting point of system execution (i.e., s_0 finishes when the system is at state q_0). s_i is the step immediately prior to s_{i+1}, and s_{i+1} is the step immediately behind s_i. Moreover, because nodes within 1-hop neighborhood executes in a mutually exclusive matter, each step takes at most $(K+1)d_0$ amount of time.

Now we define the *Containment Tree* of a node i, denoted as CT_i, which changes from a system state to another system state. We denote CT_i at the end of a step s_j as $CT_i(s_j)$, where $j = 0, 1, 2, \ldots$.

For convenience, we define the following notations:

$s_j(action_k, k')$	:	in step s_j, $action_k$ is executed at node k';
$s_j(x)$	:	the value of variable x at the end of step s_j, or the node with ID being this value;
$Anc(i, j, CT_i(s_j))$	:	node i is an ancestor of node j in tree $CT_i(s_j)$;
$Depth(CT_i(s_j))$	:	the depth of tree $CT_i(s_j)$; $Depth(CT_i(s_j)) = 0$ if $CT_i(s_j)$ is an empty tree;
$WC_i(s_j)$	:	a bookkeeping variable for $CT_i(s_j)$.

For step s_j ($j = 0, 1, \ldots$), $CT_i(s_j)$ and $WC_i(s_j)$ are defined inductively as follows:

$$
\begin{array}{rcl}
CT_i(s_0) & = & \begin{cases} G_0(V_0, E_0), where\ V_0 = \{i\}\ and\ E_0 = \emptyset & \text{If } s_0(SP.i) = true \\ G_0(V_0, E_0), where\ V_0 = E_0 = \emptyset & \text{Otherwise} \end{cases} \\
WC_i(s_0) & = & \emptyset
\end{array}
$$

and

$CT_i(s_{j+1}) = G_{j+1}(V_{j+1}, E_{j+1})$, where

$$
\begin{array}{rcl}
V_{j+1} & = & (V_j \cup V_a) \setminus V_d \\
E_{j+1} & = & (E_j \cup E_a) \setminus E_d \\
V_a & = & \{k : k \notin V_j \wedge s_{j+1}(C_1, k) \wedge \\
 & & \quad (s_j(p.k) \in V_j \vee (s_j(SP.k) \wedge (\exists k' : G.k'.k))) \wedge \\
 & & \quad \neg(\exists k'' : k' \in V_d \wedge (k'' = s_j(p.k) \vee Anc(k'', s_j(p.k), CT_i(s_j)))) \\
 & & \} \\
G.k'.k & = & k' \in V_j \wedge k' \in N.k \wedge d.k' < d.k \wedge s_j(ghost.k') \wedge (\forall l : l \in N.k \wedge s_j(ghost.l) \Rightarrow d.l \geq d.k') \\
V_d & = & \{k : k \in V_j \wedge \\
 & & \quad (((s_{j+1}(C_2, k) \vee k \in WC_i(s_j)) \wedge \\
 & & \quad \neg(\exists k' : (k' \in V_a \wedge (s_j(p.k') = k \vee Anc(k, s_j(p.k'), CT_i(s_j)))) \vee \\
 & & \quad (k' \notin WC_i(s_j) \wedge Anc(k, k', CT_i(s_j)))) \\
 & & \quad) \vee \\
 & & \quad (s_{j+1}(S_2, k) \vee s_{j+1}(SC_1, k)) \wedge s_{j+1}(p.k) \notin CT_i(s_j)) \vee \\
 & & \quad (\exists k' : Anc(k', k, CT_i(s_j)) \wedge (s_{j+1}(S_2, k') \vee s_{j+1}(SC_1, k')) \wedge s_{j+1}(p.k') \notin CT_i(s_j)) \\
 & & \quad) \\
 & & \} \\
E_a & = & \{(s_j(p.k), k) : k \in V_a \wedge s_j(p.k) \in V_j \wedge s_j(p.k) \neq k\} \cup \{(k', k) : k \in V_a \wedge k' \in V_j \wedge G.k'.k\} \\
E_d & = & \{(k, k') : (k, k') \in E_j \wedge k \in V_d\} \\
WC_i(s_{j+1}) & = & (WC_i(s_j) \setminus V_d) \cup \\
 & & \{k : \text{action } C_2 \text{ is executed at node } k \text{ in } s_{j+1} \wedge (\exists k' : (k, k') \in E_{j+1})\}
\end{array}
$$

Intuitively, if a node i is not a source of corruption propagation at system state q_0, its containment tree

CT_i is empty during any system computation. If node i is a source of corruption propagation at system state q_0, its containment tree $CT_i(s_j)$ ($j = 0, 1, \ldots$) is composed of the set of nodes that are directly (i.e., executing action C_1 because their parents have executed C_1) or indirectly (i.e., becoming sources of corruption propagation and executing action C_1 because their parents have executed action C_2 or the node with the minimum "distance value" within 1-hop neighborhood has executed C_1) involved in the containment wave initiated at node i up to step s_j of system execution.

A node k is added to set V_a (i.e., added to CT_i in S_{j+1}) if it is not in V_j, action C_1 is executed at k in step s_{j+1}, either the parent of k in s_j is in V_j or node k becomes a source of corruption propagation at the end of s_j due to the fact that a neighbor k' of k becomes involved in a containment wave (i.e., $ghost.k'$ set to $true$) in s_j and k' is the only node that makes $s_{j-1}(SP.k)$ $false$ before, and none ancestor of k in $CT_i(s_j)$ is deleted from CT_i in s_{j+1}.

A node k is added to V_d (i.e., removed from CT_i in s_{j+1}) if it is in V_j, and one of the following conditions holds:

- Either action C_2 is executed at k in s_{j+1}, or C_2 has been executed at k before s_{j+1} (i.e., $k \in WC_i(s_j)$), and there is no node k' such that k is an ancestor of k' in CT_i, and k' is either in V_a or is not in $WC_i(s_j)$.
- Action S_2 or SC_1 is executed at k in s_{j+1}, and the new parent of k at the end of s_{j+1} is not in CT_i.
- Action S_2 or SC_1 is executed at a node k' which is an ancestor of k in $CT_i(s_j)$.

A node k is added to WC_i in s_{j+1} if action C_2 is executed at k in s_{j+1} and there is no descendant k' of k that is still in CT_i. A node k is removed from WC_i in s_{j+1} if it was in WC_i before s_{j+1} (i.e., $k \in WC_i(s_j)$) and it is removed from CT_i in s_{j+1}.

Moreover, the depth of containment tree CT_i rooted at node i ($i \in V$) first keeps increasing by one in every step of system computation until reaching its maximum value, and then keeps decreasing by at least one in every step of computation until the tree becomes an empty tree. The reasons are as follows:

i. For any non-empty containment tree CT_i, either $Depth(CT_i(s_{j+1})) = Depth(CT_i(s_j)) + 1$ or $Depth(CT_i(s_{j+1})) \leq Depth(CT_i(s_j)) - 1$ when a system computation goes from s_j to s_{j+1} ($j = 0, 1, \ldots$).

   According to the definition of containment tree, its depth can increase at most one in every step when $Depth(CT_i(s_j)) \neq 0$ (i.e., $CT_i(s_j)$ is not an empty tree). Whether the depth of the tree can increase in step s_{j+1} depends on the actions that are executed within 1-hop neighborhood of those nodes that are farthest from the root node i in tree $CT_i(s_j)$. let node k be any one of the such nodes. Then k must be a leaf in $CT_i(s_j)$. And at the end of step s_j, action S_2, C_2, and SC_1 can be enabled at node k, and action C_1 can be enabled at a node k' that is either a child of k or a neighbor k with $s_j(SP.k') = true$.

   If none of actions S_2 and SC_1 is executed in s_{j+1} at node k or any of its ancestors in $CT_i(s_j)$, and if action C_1 is executed at node k' in s_{j+1} either because $s_j(SP.k') = true$ or $s_{j+1}(CW.k')$ holds (i.e., $s_{j+1}(C_1, k') \wedge \neg(\exists k'' : (k'' = k \vee Anc(k'', k, CT_i(s_j))) \wedge (s_{j+1}(S_2, k'') \vee s_{j+1}(SC_1, k''))))$,

node k' will be added to V_a in s_{j+1}, and $Depth(CT_i(s_{j+1})) = Depth(CT_i(s_j)) + 1$ holds.

If the above case does not happen to any of the farthest leaf nodes in $CT_i(s_j)$, then $Depth(CT_i(s_{j+1})) \leq Depth(CT_i(s_j)) - 1$ will hold because of the following facts:

- Actions that are executed in s_{j+1} within 1-hop neighborhood of those leaf nodes that are not the farthest leaf nodes in $CT_i(s_j)$ will not make $Depth(CT_i(s_{j+1})) \geq Depth(CT_i(s_j))$ hold.
  Let us consider any leaf node k that is not the farthest in $CT_i(s_j)$. The reason that node k is not one of the farthest leaf nodes must be that every node k' with $s_{j-1}(p.k') = k$ has executed action C_2 (but no neighbor l of k' is such that $s_j(SP.l) = true$), S_2, or SC_1 in step s_j. Then in s_{j+1}, action C_2 can be enabled at node k, and there will be no neighbor k'' of node k with $s_{j+1}(SP.k'')$ being $true$ even if action C_2 is executed at k in s_{j+1}. Action S_2 or SC_1 can be enabled at node k in s_{j+1} too, but their execution cannot increase or retain the depth of the containment tree of node i from step s_j to s_{j+1} either. Therefore, the action executed at node k cannot make $Depth(CT_i(s_{j+1})) \geq Depth(CT_i(s_j))$ hold, and this is the case for every such leaf node that is not the farthest in $CT_i(s_j)$.
- If none of actions S_2 and SC_1 is executed in s_{j+1} at any ancestor of a farthest leaf node k in $CT_i(s_j)$, then either action C_2 is executed at node k in s_{j+1} but $s_j(SP.k')$ is $false$ for every neighbor k' of k, or action S_2 or SC_1 is executed at node k in s_{j+1}. Then node k is added to V_d in s_{j+1}, and therefore $k \notin CT_i(s_{j+1})$. In this case, $Depth(CT_i(s_{j+1})) = Depth(CT_i(s_j)) - 1$ holds.
- If action S_2 or SC_1 is executed in s_{j+1} at an ancestor k' of node k in $CT_i(s_j)$, then the subtree rooted at node k' in $CT_i(s_j)$ will be removed from the containment tree of node i in s_{j+1}. If this happens to all the farthest leaf nodes, then $Depth(CT_i(s_{j+1})) < Depth(CT_i(s_j)) - 1$ will hold because the minimum depth of these subtrees is 2.

ii. If there is only one non-empty containment tree CT_i in the system, once the depth of CT_i decreases in a step s_j ($j = 1, 2, \ldots$), it keeps decreasing in every step later on until reaching 0.

If there is only one non-empty containment tree CT_i in the system, the depth of the tree will not increase because of actions executed at the 1-hop neighborhood of an intermediate node m in the tree. This is because no neighbor m' of m that is not in CT_i will become a child of m later on due to the fact that $ghost.m$ is $true$ and action S_2 will not be enabled at node m'.

Let us consider any leaf node k in $CT_i(s_j)$. If node k is removed from CT_i in s_j and node l ($l = s_{j-1}(p.k)$) becomes a leaf in s_j, then node l will be removed from CT_i in s_{j+1}. The reason is as follows. If node k executes action C_2 in s_j but no child k' of k is such that $s_j(SP.k') = true$, or if node k executes action S_2 or SC_1 in s_j, then node k will be removed from CT_i in s_j (i.e., $k \notin CT_i(s_j)$). Therefore node l will be a leaf node in $CT_i(s_j)$, if $l \neq k$ and k is the only child of l in s_{j-1}, or the same case that has happened to k happens to every other node m with $s_{j-1}(p.m) = l$. If node l does become a leaf of $CT_i(s_j)$, it will be removed from tree CT_i in step s_{j+1} because: in s_{j+1}, action C_2 can be enabled at node l, and there will be no child k'' of l with $s_j(SP.k'')$ being $true$ even if action C_2 is executed at l in s_{j+1}; action S_2 or SC_1 can also be enabled at node k in

s_{j+1}, but their execution only removes node k from CT_i.

If the depth of the containment tree CT_i decreases in a step s_j, then every leaf node k that is one of the farthest leaf nodes in $CT_i(s_{j-1})$ must have executed action C_2 (but no node k' with $s_{j-1}(p.k') = k$ is such that $s_{j-1}(SP.k') = true$), S_2, or SC_1, and so does any other leaf node that is not the farthest in $CT_i(s_{j-1})$. Therefore, every leaf node in $CT_i(s_j)$ will be removed from CT_i in step s_{j+1} according to the analysis above. Thus, the depth of CT_i will decrease in step s_{j+1}, and so on until it reaches 0.

iii. The above two properties of a containment tree hold even if there are more than one non-empty containment trees in the system. That is, two different containment trees do not interfere with each other in the sense that the two properties hold in the presence of multiple non-empty containment trees.

Since the proof for the first property does not depend on the assumption that there is only one non-empty containment tree in the system, we only consider the second property here.

Without loss of generality, let us consider any two containment trees CT_i and $CT_{i'}$. In a system computation, if there exists no two nodes k and k' such that k and k' are neighbors, $k \in CT_i$ in a set of steps R, and $k' \in CT_{i'}$ in a set of steps R', the above claim trivially holds. If there exists two nodes k and k' such that k and k' are neighbors, $k \in CT_i$ in a set of steps R, and $k' \in CT_{i'}$ in a set of steps R', the above claim still holds because of the following facts:

- If there exists a step s_j ($j = 0, 1, \ldots$) such that $s_j \in R$ and $s_j \in R'$, then $s_j(ghost.k)$ and $s_j(ghost.k')$ are $true$. Therefore, action S_2 and SC_1 that can set k' as the parent of k will not be executed at k in s_{j+1}, neither will action S_2 and SC_1 that can set k as the parent of k' be executed at k' in s_{j+1}. Therefore, there will be no interference between CT_i and $CT_{i'}$ in s_{j+1}.
- If there exists no step s_j such that $s_j \in R$ and $s_j \in R'$, but exists a step s_l ($l = 0, 1, \ldots$) such that $s_l \in R \wedge s_l \notin R'$ or $s_l \notin R \wedge s_l \in R'$, then either $s_j(ghost.k)$ or $s_j(ghost.k')$ is $true$, and neither S_2 nor SC_1 that can set k' as a child of k or set k as a child of k' will be executed at k' or k in s_{l+1}. Therefore, there will be no interference between CT_i and $CT_{i'}$ in s_{l+1}.
- If there exists no step s_j such that $(s_j \in R \wedge s_j \in R') \vee (s_j \in R \wedge s_j \notin R') \vee (s_j \notin R \wedge s_j \in R')$, then the claim trivially holds because there is no way for the two containment trees to interfere with each other.

Now, we are ready to prove the stabilization of the system from state q_0 to a state in $F.0'$ in $O(N)$ time. If $F.0'$ holds at q_0, the claim trivially holds. So, we only need to prove that the system converges from a state q_0 in $\neg F.0'$ to a state in $F.0'$ in $O(N)$ time.

We prove this by exhibiting a variant function [2] $f(s_j)$ ($j = 0, 1, \ldots$) as follows:

$$
\begin{aligned}
f(s_j) \;&=\; \langle G(s_j), T(s_j)\rangle, \text{ where} \\
&\qquad G(s_j) \;=\; |\,\{i : s_j(ghost.i) \;\wedge\; \neg(\exists k : k \in V \wedge i \in CT_k(s_j))\}\,| \\
&\qquad T(s_j) \;=\; |\,\{i : CT_i(s_j) \text{is a non-empty tree}\}\,|
\end{aligned}
$$

Function $f(s_j)$ is non-increasing as j increases, and it reaches its minimum value $\langle 0, 0\rangle$ within $O(N)$ amount of time. This is because the following facts:

- Function $G(s_j)$ is non-increasing as j increases. And it decreases to its minimum value 0 within $O(N)$ amount of time.

  Among all the actions, only the execution of action C_1 at an arbitrary node i can turn $ghost.i$ from $false$ to $true$. But the execution of C_1 at a node i adds i into a containment tree at the same time. Therefore, function $G(s_j)$ is non-increasing as j increases.

  If $G(s_j)$ is not equal to 0 at the end of any step s_j, it must decrease by at least one in s_{j+1}. The reason is as follows: if $G(s_j) \neq 0$, there must exist at least one node i where $ghost.i = true$ and action S_2, C_2, SC_1, or SC_2 is enabled at the end of s_j; then, the execution of any of these actions at node i in s_{j+1} will set $ghost.i$ to $false$.

  Since $0 \leq G(s_j) \leq N$, $G(s_j)$ will reach its minimum value in no more than N steps. Because each step takes at most $(K+1)d_0$ amount of time, $G(s_j)$ reaches its minimum value within $(K+1)d_0N$, i.e., $O(N)$, amount of time.

- Function $T(s_j)$ is non-increasing as j increases.And it decreases to its minimum value 0 within $O(N)$ amount of time.

  According to the definition of containment tree, the containment tree CT_i of a node i is non-empty only if $s_0(SP.i) = true$, and an empty containment tree remains empty in a system computation. Therefore, function $T(s_j)$ is non-increasing as j increases, and the number of non-empty containment trees in a system is non-increasing.

  Since the depth of any non-empty containment tree first keeps increasing by one in every step of system computation until reaching its maximum value, and the maximum depth of any containment tree is N, every containment tree will reach its maximum depth within N steps, i.e., within $O(N)$ amount of time. After reaching its maximum value, the depth of a containment tree keeps decreasing by at least one in every step of computation until it reaches the minimum value 0, and this decreasing procedure takes no more than N steps of computation, thus takes $O(N)$ amount of time. Therefore, every containment tree will become an empty tree within $O(N)$ amount of time.

  Moreover, there is no interference between any two containment trees in their growing or shrinking procedure. Thus, all the containment trees will become empty, and $T(s_j)$ reaches its minimum value 0 within $O(N)$ amount of time.

When the value of $f(s_j)$ is $\langle 0, 0\rangle$, there exists no node i where $ghost.i = true$ or $SP.i = true$. Therefore, starting at state q_0, the system reaches a state in $F.0'$ within $O(N)$ amount of time.

(b) $F.0'$ is closed under system execution.

In order to prove the closure of $F.0'$ under system execution, we need to prove that every action at every node in the system preserves $F.0'$. Without loss of generality, let us consider an arbitrary node i in the system. Since actions C_1, C_2, SC_1, and SC_2 are not enabled at node i at every state in $F.0'$, they trivially preserve $F.0'$. Also, because action S_1 does not modify the value of $d.i$ and $ghost.i$, it trivially preserve

$F.0'$ too. Action S_2 at node i preserves $F.0'$ due to the following facts:

- After the execution of action S_2 at node i, $ghost.i$ is set to $false$, and there exists a neighbor j with $d.j = d.i - 1$. Thus, both $ghost.i$ and $SP.i$ are equal to $false$;
- For any neighbor k of i, if $SP.k = false$ because $d.i < d.j$ before the execution of S_2 at i, then $d.i < d.j$ still holds after the execution since $d.i$ decreases. Thus $SP.k$ still equals to $false$ after the execution of S_2 at node i. Moreover, the value of $ghost.k$ remains unchanged after the execution of S_2 at i.

(c) $F.0'$ implies $F.0$.

In order to prove that $F.0'$ implies $F.0$, we only need to prove that $F.0'$ implies $(\forall i : i \in V \Rightarrow d.i \geq 0)$. This is proved by contradiction. Let us assume that there is a node i with $d.i < 0$ at a system state q'_0 in $F.0'$. If $i = r$, then $SP.i = true$ at state q'_0, which contradicts the fact that q'_0 is a state in $F.0'$; If $i \neq r$ and $SP.i = false$ at state q'_0, then there must exists another node i' with $d.i' < 0$ and $SP.i' = false$ at state q'_0, which also contradicts the fact that q'_0 is a state in $F.0'$. Therefore, $F.0'$ implies $F.0$.

(II) For each l such that $0 \leq l \leq D$: $F.l$ is closed under system execution; that is, once $F.l$ holds in an arbitrary system computation, it continues to hold subsequently.

The proof for the closure of $F.0$ under system execution follows exactly from that for the closure of $F.0$ under system execution as we have shown before, thus we neglect it here.

Now we prove the closure of $F.l$ under system execution for any l that is larger than 0. For any l ($l \geq 1$), actions C_1, C_2, SC_1, and SC_2 are enabled at no node in the system at a state in $F.l$, because $(\forall i : i \in V \Rightarrow \neg ghost.i \wedge \neg SP.i)$ holds at any state in $F.l$. Moreover, actions S_1 and S_2 are disabled at every node that is no more than $(l-1)$-hop away from node r at a state in $F.l$. Even though actions S_2 can be enabled at a node that is l-hop away from r at a state in $F.l$, its execution does not falsify $LH.i$ of any node i that is no more than $(l-1)$-hop away from r. Therefore, $F.l$ is closed under system execution for any l that is larger than 0.

(III) For each l such that $0 \leq l < D$: upon starting at an arbitrary state in $F.l$, the system is guaranteed to reach a state in $F.(l+1)$ within a constant amount of time.

Since $F.l$ is closed under system execution, and $F.(l+1)$ is defined as $F.l \wedge (\forall i : i \in V \wedge dist(i, r, G) = l \Rightarrow LH.i) \wedge (\forall i : i \in V \wedge dist(i, r, G) > l \Rightarrow d.i > l)$, we only need to prove that starting at an arbitrary state in $F.l$, the system is guaranteed to reach a state where $(\forall i : i \in V \wedge dist(i, r, G) = l \Rightarrow LH.i) \wedge (\forall i : i \in V \wedge dist(i, r, G) > l \Rightarrow d.i > l)$ holds. This is proved in two steps:

- First, starting at an arbitrary state in $F.l$, the system is guaranteed to reach a state where $(\forall i : i \in V \wedge dist(i, r, G) = l \Rightarrow LH.i)$ holds.

  If there exists any node i where $dist(i, r, G) = l$ and $LH.i = false$, then action S_2 must be enabled at node i, and the execution of S_2 at i will make $LH.i$ become $true$, which needs only one step of system computation.

- Second, $F.l \wedge (\forall i : i \in V \wedge dist(i, r, G) = l \Rightarrow LH.i)$ implies $(\forall i : i \in V \wedge dist(i, r, G) > l \Rightarrow d.i > l)$.

This trivially holds. Otherwise, there would be a node i that is no less than $(l+1)$-hop away from node r and $SP.i$ is $true$, which violates $F.l$.

(IV) $F.D \equiv F$

This trivially holds because the maximum distance between a node i ($i \neq r$) and r is $D-1$.

Based on the analysis above, we can see that starting at an arbitrary state, the system is guaranteed to reach a state in F within $O(N+D) = O(N)$ amount of time.

□

Corollary 1 (Termination of the algorithm) *Starting at an arbitrary state, the system computation terminates at a state in F within $O(N)$ amount of time, where N is the number of nodes in the system.*

Proof: This claim follows from Theorem 2 and the fact (by Theorem 1) that no action is enabled in the system at a state in F.

□

From Corollary 1, we know that our algorithm guarantees the formation of the shortest path tree within finite amount of time after starting at an arbitrary state.

Now, let us prove the properties of local containment and stabilization of our algorithm, by which the shortest path tree is guaranteed to be formed within time proportional to the perturbation size at the initial state, and the range of contamination during stabilization is also proportional to the perturbation size.

Lemma 1 (Local containment and stabilization of a single perturbed region) *Starting at an arbitrary state q_0 where the set of perturbed nodes are contiguous, the system G self-stabilizes to a state in F within $O(P(G,q_0))$ amount of time, and the range of contamination during stabilization is $O(P(G,q_0))$.*

Proof: Let K be the maximal degree of nodes in a system $G(V,E)$, and p be the perturbation size $P(G,q_0)$ for the system at state q_0. Also, we assume that the time taken to execute any action is bounded from above by d_0, and bounded from below by d'_0.

Let R_p be the perturbed region, and p_d be the diameter of the subgraph of G on R_p. The set of nodes that are not in R_p is called the *healthy region* R_h[6]. A node i in R_p is called a *perturbed boundary node* if it has a neighbor j that is in R_h, and node j is called a *healthy boundary node* correspondingly.

There are three variables (i.e., $d.i$, $p.i$, and $ghost.i$) maintained at each node i in the system. According to the algorithm, the corruption of $p.i$ at node i does not affect any neighbor j of i, therefore the corruption of $p.i$ at any node i does not propagate. However, the corruption of $d.i$ or $ghost.i$ at i can affect the neighbors of i, thus it can propagate. Since the corruption of $d.i$ is more complicated (we can see this later in the proof) than that of $ghost.i$, we use it as the major theme leading the proof here.

When a single perturbed region occurs in the system, there can be two different cases:

[6]The subgraph of G on R_h can be disconnected.

(I) Case one: the "distance value" of every perturbed node is no less than that of all the healthy boundary nodes, i.e., $(\forall i, j : i \in R_p \wedge j$ is a healthy boundary node $\Rightarrow d.i \geq d.j)$.

Let us first consider the case where there exists at least one perturbed node l where $d.l$ is corrupted at state q_0. Then, it is possible that there exists no healthy boundary node that is a source of corruption propagation at state q_0; it is also possible that there exist some healthy boundary nodes that are sources of corruption propagation at q_0.

(a) If no healthy boundary node is a source of corruption propagation at state q_0, then every healthy boundary node i must have a healthy parent at state q_0, because no perturbed node can be the parent of node i due to the fact that $(\forall j : j \in R_p \Rightarrow d.j \geq d.i)$ in this case. Therefore, the perturbed region R_p looks like a "big leaf node" in the minimum distance routing tree (as shown in Figure 3).

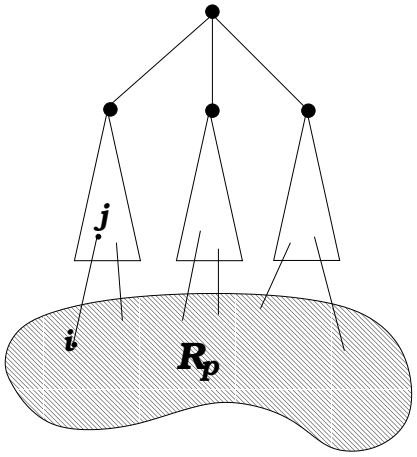


Figure 3: The "distance value" of every perturbed node is no less than that of all the healthy boundary nodes, and there is no source of corruption propagation.

In this case, no action is enabled at any node in R_h, thus no action will be executed at any node in R_h. Therefore, the range of contamination during stabilization is 0, i.e., $R_c(G, q_0) = 0$, and $R_c(G, q_0) = O(p)$ trivially holds.

Since no action is executed at any node in R_h, proof for the convergence of the set of nodes in R_c to a legitimate state follows that for Theorem 2. As shown in the proof for Theorem 2, the number of non-empty containment trees (defined in the proof for Theorem 2) reaches 0 within $O(p)$ amount of time in this case. When all the containment trees are empty, the system is at a state q_0' where $(\forall k : k \in V \Rightarrow \neg ghost.i \wedge \neg SP.k \wedge d.k \geq 0)$ holds.

Starting at state q_0', only action S_2 is enabled at any perturbed boundary node, and it is continuously enabled at every perturbed boundary node until it is executed. Therefore, in a step of system computation, action S_2 is executed at every perturbed boundary node i, making $LH.i$ equal to $true$, and every such node i is first "deleted" from R_p and then "added" to R_h. Therefore, the perturbed region R_p keeps shrinking until it becomes empty, while the healthy region R_h keeps expanding until it includes all the nodes in the

system.

Let us define a function $f = max\{dist(i, R_h) : i \in R_p\}$, where $dist(i, R_h) = min\{dist(i, j, G) : j \in R_h\}$ and $dist(i, j, G) =$ minimum distance between node i and j in graph G. By its definition, $0 \leq f \leq p_d$ holds. Also, f decreases by one in every step of system computation. Therefore, starting at state q'_0, function f reaches its minimum value 0 within $O(p_d)$ steps of system computation. When $f = 0$, there is no perturbed node any more and the system is at a state in F. Because each step of computation takes at most time $(K + 1)d_0$ amount of time, the system reaches a state in F within $O(p_d)(K + 1)d_0 = O(p_d)$ amount of time.

Therefore, the system reaches a state in F within $O(p) + O(p_d) = O(p)$ amount of time.

(b) If there are some non-root[7] healthy boundary nodes that are sources of corruption propagation at state q_0, containment waves can be initiated at those nodes, and the perturbed region R_p can expand outward to the healthy region R_h. But this expansion is bounded.

Let us first consider the case where there is only one such non-root healthy boundary node i that is a source of corruption propagation at q_0 (see Figure 4[8]). The parent of node i must be perturbed at q_0 (i.e., $p.i \in R_p$) because otherwise node i would not be a source of corruption propagation. Action C_1 is the only action that is enabled at node i at q_0, and the execution of C_1 at i will initiate a containment wave that is propagated toward the subtree rooted at node i, by the execution of action C_1 at every node involved in this containment wave. Thus, the perturbed region R_p expands outward to the healthy region R_h. However, this containment wave will not propagate unbounded because of the following facts:

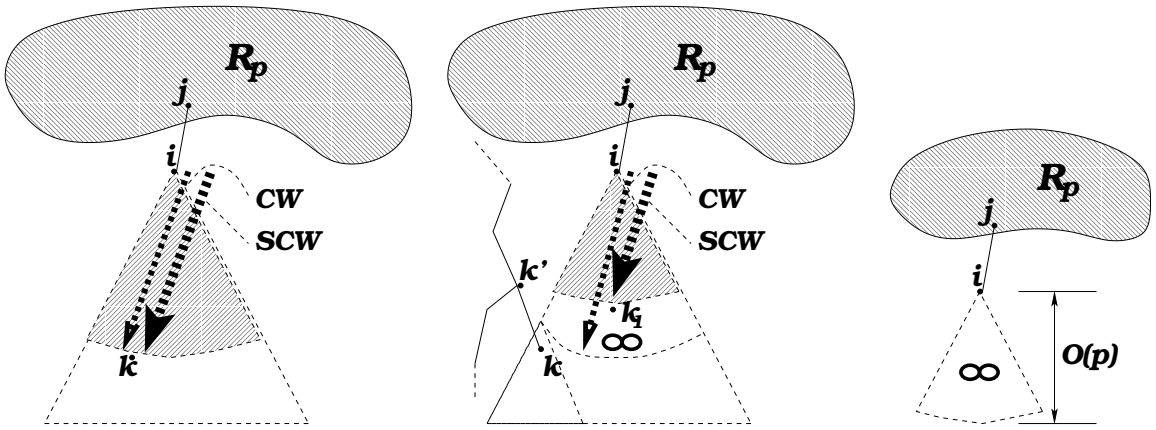


Figure 4: The "distance value" of every perturbed node is no less than that of all the healthy boundary nodes, and there are some sources of corruption propagation.

[7]The root node r cannot be a healthy boundary node that initiates a containment wave, because $SP.r = false$ when $ghost.r = false$ and $LH.r = true$.

[8]In the figures of this proof, SW, CW, and SCW stand for stabilization wave, containment wave, and super-containment wave respectively.

i. If action C_2 has not been executed at node i before the moment that is one step after all the nodes in S, the original set of perturbed nodes at state q_0, have converged to a legitimate state (see Figure 4(a)(b)), the farthest distance (in terms of hops) the containment wave can propagate is $O(p)$, and the system reaches a state in F in $O(p)$ amount of time. The reasoning is as follows.

A. Starting at state q_0, it takes $O(p)$ amount of time for all the nodes in S to converge to a legitimate state.

As in the proof for Ia above, we first show that starting at state q_0, the system reaches a state $q_0'$where $(\forall k : k \in V \Rightarrow \neg ghost.i \wedge \neg SP.k \wedge d.k \geq 0)$ holds within $O(p)$ amount of time. The reason is as follows. Because the "distance value" of every perturbed node is no less than that of all healthy boundary nodes, no action except C_1 can be enabled at a healthy boundary node i' that is not a source of corruption propagation at q_0, but the execution of C_1 at i' can only add i' to the containment tree rooted at another healthy boundary node that is a source of corruption propagation at state q_0 (in this case, it is node i). As for node i, no action except C_1 can be enabled at it. But the execution of C_1 at i only adds node i to a containment tree rooted at itself. Before action C_2 is executed at node i, actions S_2 and SC_1 that can set i as the parent of its neighbor j in S will not be executed at j because $ghost.i = true$. Therefore, no node in S will be involved in the containment wave initiated at i, and actions executed at nodes in $(V \setminus S)$ will not affect actions executed at nodes in S in terms of the growing or shrinking of containment trees in S. Thus, the proof follows that for Theorem 2 too. As shown in the proof for Theorem 2, the number of non-empty containment trees reaches 0 within $O(p)$ amount of time. When all the containment trees are empty, the system is at a state q_0' where $(\forall k : k \in V \Rightarrow \neg ghost.i \wedge \neg SP.k \wedge d.k \geq 0)$ holds.

In exactly the same way as in Ia, we can see that, starting at state q_0', the set of nodes in S converge to a legitimate state within $O(p_d)$ amount of time. Let us denote the system state at this moment as state q_1.

Therefore, starting at state q_0, the set of nodes in S converge to a legitimate state within $O(p) + O(p_d) = O(p)$ amount of time.

B. After all the nodes in S converge to a legitimate state q_1, the system stabilizes to a state in F within $O(p)$ amount of time.

When all the nodes in S converge to a legitimate state q_1, and action C_2 has not been executed at node i, either action S_2 or action SC_1 will be executed at i. And a super-containment wave will propagate from i downward to nodes in the containment tree CT_i, by the execution of action SC_2 at those nodes. There are two possible cases here:

- Action C_2 has not been executed at any node j in CT_i before the super-containment wave arrives at it (see Figure 4(a)).

  In this case, because the super-containment wave propagates faster than the containment wave, it will catch up with the containment wave, and sets the state of every node l in CT_i to a legitimate one where $\neg ghost.l \wedge LH.l$ holds. Let p_{cw} be the maximum distance (in terms of

the number of hops) that the containment wave can propagate before the super-containment wave catches up with it. Then $(k \cdot d_0 + d_{sc})p_{cw} + O(p) = (d'_0 + d_c)p_{cw}$ holds, which implies that $p_{cw} = \frac{O(p)}{d_c - d_{sc} + d'_0 - k \cdot d_0} = O(p)$.[9]

Therefore, the range of contamination and the time taken for the system to stabilize to a state in F is $O(p)$ in this case.

- Action C_2 has already been executed at some node k_1 in CT_i before the super-containment wave arrives at it (see Figure 4(b)).

  In two cases will this happen: first, before the super-containment wave arrives at k_1, the containment wave has reached the leaf nodes of the subtree rooted at k_1 in CT_i; second, there exists a node k in the subtree rooted at k_1 in CT_i such that $(\exists k' : k' \in N.k \wedge k' \notin CT_i \wedge PS.k.k')$.

  It is easy to see that in either case, the range of contamination is no more than that in the case where action C_2 has not been executed at any node j in CT_i before the super-containment wave arrives at it. Therefore, the range of contamination is still $O(p)$.

  For any node l in CT_i where action C_2 has been executed before the super-containment wave reaches it, action S_2 will be executed at it within time proportional to $min\{dist(l, m, G) : m \in CT_i \wedge$ the super-containment wave has reached node $m\}$ that is $O(p)$. And the execution of S_2 at l will set $LH.l$ to $true$.

  Following the same reasoning as in Ia, we can see that the system stabilizes to a state in F within time $O(p)$ in this case.

ii. If action C_2 has been executed at node i before the moment that is one step after all the nodes in S (i.e., the original set of perturbed nodes at state q_0) have converged to a legitimate state (see Figure 4(c)), the farthest distance the containment wave can propagate is still $O(p)$, and the system reaches a state in F in $O(p)$ amount of time. The reasoning is as follows.

  It is easy to see that the depth of the containment tree CT_i is $O(p)$ in this case. Thus the range of contamination is also $O(p)$.

  After the execution of C_2 at node i, $d.l = \infty$ for every node l that was in CT_i once and has executed C_2. Let us denote the set of these nodes as S'. It is possible for some nodes in S' to be added to the containment tree of some node j' in S, but this can increase the depth of $CT_{j'}$ by at most $O(p)$. Therefore, the system will still reach a state q'_0 where all the containment trees are empty within time $O(p)$. And the same as analyzed before, the system will reach a state in F within time $O(p)$, starting at state q'_0. Therefore, starting at state q'_0, the system reaches a state in F within time $O(p)$.

If there exist multiple healthy boundary nodes that are sources of corruption propagation at state q_0, the same result still holds, due to the fact that there is no interference among stabilization waves, containment waves, and super-containment waves that are initiated at different nodes in the sense that the range of contamination is retained for the system during stabilization[10].

[9] $d_c - d_{sc}$ should be larger than $k \cdot d_0 - d'_0$.

[10] The reasoning is similar to that in the proof for Theorem 2. For simplicity, we neglect it here

If there exists no node l where $d.l$ is corrupted at state q_0, then the case is the similar to that as shown in Figure 3 in the sense that the range of contamination is 0. And following the same analysis as in the proof for Theorem 2, the system stabilize to a state in F in $O(p)$ amount of time.

(II) Case two: the "distance value" of some or all perturbed nodes is less than that of some or all healthy boundary nodes.

Again, let us first consider the case where there exists at least one perturbed node l where $d.l$ is corrupted at state q_0. Then, it is possible that there exists no healthy boundary node that is a source of corruption propagation at state q_0; it is also possible that there exist some healthy boundary nodes that are sources of corruption propagation at q_0. Again, we let S be the set of perturbed nodes at state q_0.

(a) If no healthy boundary node is a source of corruption propagation at state q_0 (see Figure 5), then the containment wave will not be initiated at any healthy boundary node at system state q_0. But action S_2 can be enabled, and the stabilization wave be initiated at a healthy boundary node i either at state q_0 or at some state during stabilization, because there can exist a neighbor j of i in R_p such that $d.j < d.i - 1$ due to the fact that the distance value of some perturbed node in R_p can be less than $(d.i - 1)$ either at state q_0 or at some state during stabilization.

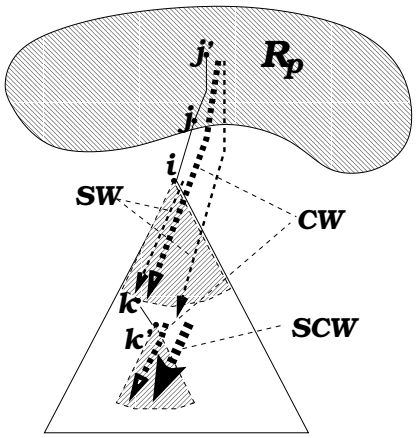


Figure 5: The "distance value" of some or all perturbed nodes is less than that of some or all healthy boundary nodes, and there is no source of corruption propagation.

Same as in Ib, let us first consider the case where there is only one such healthy boundary node i that can initiate a stabilization wave at state q_0 or at some state during stabilization.

Even though a stabilization wave can be initiated at node i, it will not propagate unbounded due to the fact as follows. If there exists a neighbor j of i such that $d.j < d.i - 1$ at some state, and action S_2 is executed at i which sets j as the parent of i, then there must exist a node j' in S where $SP.j' = true$ at state q_0, and both node i and j are added to the containment tree $CT_{j'}$ rooted at j'. Since the distance between i and j' is $O(p)$, the containment tree $CT_{j'}$ will grow to reach node i no later than $O(p)$ amount of time after i initiated the stabilization wave. Then, by executing action C_1, node i will propagate the

containment wave (when $CW.i = true$) or initiate a new containment wave (when $SP.i = true$) to the set of nodes S_i that have been involved in the stabilization wave initiated at i. And node i as well as all the nodes in S_i will be added to the containment tree $CT_{j'}$. Because the containment wave propagates faster than the stabilization wave, it will catch up with and stop the propagation of the stabilization wave. Let p_{sw} be the maximum distance that the stabilization wave initiated at node i can propagate before the containment wave catches up with it. Then $(k \cdot d_0 + d_c)p_{sw} + O(p) = (d'_0 + d_s)p_{sw}$ holds, which implies that $p_{sw} = \frac{O(p)}{d_s - d_c + d'_0 - k \cdot d_0} = O(p)$.[11] Same as in I(b)i, the stabilization wave may have already stopped propagating before the containment wave catches up with it, but the distance the stabilization wave has propagated is still $O(p)$ (for simplicity, we do not discuss this situation any more).

Let k is any node where the containment wave catches up with the stabilization wave. After action C_2 is executed at node k (k will not be r) and $d.k$ is set to ∞, its neighbor k' (if any) can become a source of corruption propagation. Thus k' can initiate a containment wave by executing action C_1. But this containment wave will not propagate unbounded because of the fact as follows. Since the depth of the tree $CT_{j'}$ increases only by $O(p)$ due to the stabilization wave and containment wave initiated (or propagated) at node i, all the nodes in S will still converge to a state q'_0 where all the containment trees rooted at a node in S are empty (as proved in the proof for Theorem 2). Starting at state q'_0, all the nodes in S will converge to a legitimate state q_1 within $O(p_d)$ amount of time. Starting at state q_1, node k will execute action S_2 within $O(p)$ amount of time, and reaches a legitimate state where no action is enabled any more. Then action S_2 or SC_1 will be executed at node k', which initiates a super-containment wave downward the tree $CT_{k'}$. The super-containment wave will catch up with the containment wave within $O(p)$ amount of time, and the distance that the containment wave initiated at k' has propagated is also $O(p)$[12].

Same as that in Ib, if there are multiple healthy boundary nodes that are not sources of corruption propagation but initiate stabilization waves, the same claim hold.

Therefore, both the range of contamination and the time taken to for the system to stabilize to a state in F are $O(p)$ in this case.

(b) If there are some non-root healthy boundary nodes that are sources of corruption propagation at state q_0, the claim still holds.

Same as in IIa, here we only consider the case where there is only one healthy boundary node i that is a source of corruption propagation at state q_0, and the reasoning carries to the case where there are multiple healthy boundary nodes that are sources of corruption propagation at q_0.

Since node i is a source of corruption propagation at state q_0, it can initiate a containment wave downward to its descendants in the routing tree at q_0. But the containment wave will not propagate unbounded because of the following facts:

i. If there exists no neighbor j of i in R_p such that $d.j < d.i - 1$ holds before all the nodes in S converge to a legitimate state (see Figure 6(a)), then node i will initiate a super-containment wave after all the

[11] $d_s - d_c$ should be larger than $k \cdot d_0 - d'_0$.
[12] The reasoning is the same as that in Ib

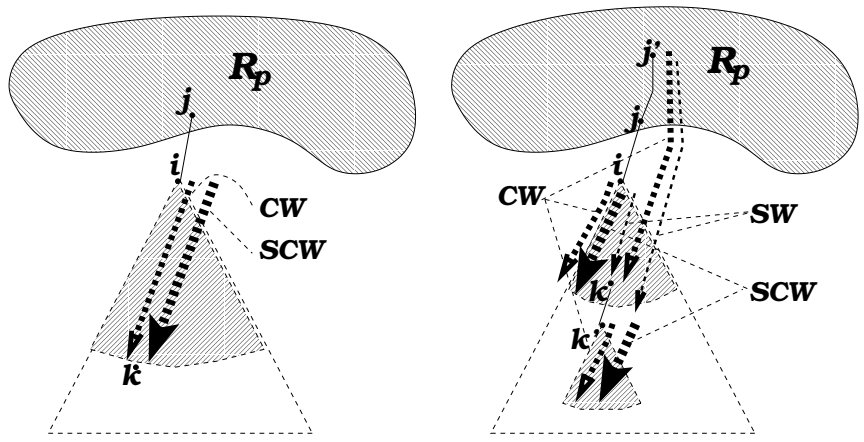


Figure 6: The "distance value" of some or all perturbed nodes is less than that of some or all healthy boundary nodes, and there are some sources of corruption propagation.

nodes in S have converged to a legitimate state. This case is the same as that in Ib, thus the claim holds.

ii. If there exists a neighbor j of i in R_p such that $d.j < d.i - 1$ before all the nodes in S converge to a legitimate state (see Figure 6(b)), then action S_2 or SC_1 will be executed at node i, and a stabilization wave be initiated at i. After action S_2 or SC_1 is executed at i, a super-containment wave will also be initiated at i, and it propagates downward the containment tree CT_i earlier as well as faster than the stabilization wave initiated at i after the execution of S_2 or SC_1 at i. As shown before, the super-containment wave will catch up with the containment wave and stop its propagation within $O(p)$ amount of time. After the stabilization wave is initiated at node i, the case is the same as that in IIa, thus the claim holds.

In this case, there can also exist some healthy boundary nodes that are not sources of corruption propagation but initiate stabilization waves as discussed in IIa. Since there is no interference between stabilization waves, containment waves, and super-containment waves initiated at different healthy nodes, the claim still holds in this case.

If there exists no node l where $d.l$ is corrupted at state q_0, then the case is similar to that as shown in Figure 3 in the sense that node i can propagate a containment wave initiated at a node j' in R_p, but the containment wave will not propagate unbounded because a super-containment wave initiated at a node j'' (j'' can be the same as j') in R_p will catch up with the containment wave and stop it. Therefore, the claim trivially holds in this case.

According to the analysis above, Lemma 1 holds.

□

Lemma 2 (Local containment and stabilization of multiple perturbed regions) *Starting at an arbitrary state q_0 where the set of perturbed nodes are non-contiguous, the system G self-stabilizes to a state in F within $O(P(G, q_0))$ amount of time, and the range of contamination during stabilization is $O(P(G, q_0))$.*

Proof: Similar to the definition of containment tree for each node that is a source of corruption propagation as in the proof for Theorem 2, we can define the *Contamination Region* for each perturbed region S at state q_0(denoted as CR_s) as the union of S and the set of nodes that are healthy at state q_0 but have been involved in some stabilization wave, containment wave, or super-containment wave initiated due to the existence of S, as described in the proof for Lemma 1.

Now let us consider any two perturbed regions S and S' at state q_0. If there exist no two neighboring nodes k and k' such that $k \in CR_S$ and $k' \in CR_{S'}$, it trivially follows that there exists no interference between S and S' during stabilization in the sense that actions being executed at nodes in one contamination region do not affect actions being executed in another contamination region. Even if there exists two neighboring nodes k and k' such that $k \in CR_S$ and $k' \in CR_{S'}$, there is still no interference between the two contamination regions in the sense that the size of the contamination region of a perturbed region in the presence of another adjourning contamination region can be no larger than that in the absence of the adjourning contamination region, since there is no interference among stabilization waves, containment waves, and super-containment waves that are initiated at different nodes in the sense that the range of contamination is retained for the system during stabilization [13].

Therefore, if there exist multiple perturbed regions $S_0, S_1, \ldots, S_m$ in the system at state q_0, the range of contamination as well as the time taken for the system to stabilize to a state in F depends on that of the perturbed region where the range of contamination is the largest or the time taken for all the nodes in its contamination region to converge to a legitimate state is the longest, due to the fact that actions are executed in parallel at different contamination regions and there is no interference between different contamination regions in the sense that the range of contamination is at least retained if not reduced. That is, the range of contamination as well as the time taken for the system to stabilize to a state in F is $max\{O(P(S_i, q_0) : i = 0, 1, \ldots, m\} = O(P(G, q_0))$.

□

Theorem 3 (Local containment and stabilization of the system) *Starting at an arbitrary state q_0, the system G self-stabilizes to a state in F within $O(P(G, q_0))$ amount of time, and the range of contamination during stabilization is $O(P(G, q_0))$. That is, the system is $\mathcal{F}$-local containing as well as $\mathcal{F}$-local stabilizing, where $\mathcal{F}$ is a linear function.*

Proof: This theorem follows from Lemma 1 and Lemma 2.

□

By Theorem 3, we can see that our algorithm satisfies the requirements of local containment and local stabilization in minimum distance routing as specified in Section 2.3.

[13] The reasoning is similar to that in the proof for Theorem 2. For simplicity, we neglect it here.

4.2 Properties of loop freedom and fast loop removal

Loop freedom is important in network routing in terms of network resource utilization and quality in communication. Here we prove the properties of loop freedom and fast loop removal of our algorithm for minimum distance routing.

Theorem 4 (Loop Freedom) *There exists no loop in the system during stabilization, if there is no loop in the system at the initial state.*

Proof: We prove this claim by contradiction. Let q_0 be the initial system state when the system computation starts. And let us assume, on the contrary to the theorem, that there exists a set of nodes L that are in a loop of length [14] l ($l \neq 0$) at a system state q_1 other than q_0 (i.e., $q_1 \neq q_0$). We denote the set of nodes in the loop as $v_0, v_1, \ldots, v_{l-1}$ with $p.v_i = v_{i-1}$ ($i = 0, 1, \ldots, l-1$)[15] at state q_1.

Then the loop L must have been in the system at the state q_1' that is 1-step prior to state q_1 (i.e., q_1 is a state of the system after a step of system computation that starts at state q_1' finishes). The reason is as follows (by contradiction).

- If loop L does not exist in the system at state q_1', then there must exist a node v_i in L such that $p.v_i \neq v_{i-1}$ at state q_1', but $p.v_i = v_{i-1}$ at state q_1. Then either action S_2 or action SC_1 is executed at node i in the step of computation between state q_1' and q_1, which implies that $d.v_{i-1} < d.v_i$ at state q_1'.

- Let $S = \{k : k \in L \wedge$ either action S_2 or action SC_1 is executed at node k in the step of computation between state q_1' and $q_1\}$, and $S' = L \setminus S$. As shown above, for every node k in S, $d.v_{k-1} < d.v_k$ at state q_1'. Now we show that for every node v_j in S', $d.v_{j-1} \leq d.v_j$ at state q_1'.

  Again, we prove this by contradiction. If $d.v_{j-1} \leq d.v_j$ does not hold at state q_1', then $d.v_{j-1} > d.v_j$ holds at q_1'. There are two possible cases here:

  - If node v_j is a minimal point or a source of corruption propagation at q_1' (i.e., $MP.v_j \vee SP.v_j = true$ at q_1'), then either action S_1 or action C_1 is executed at node v_j in the step of computation between q_1' and q_1, and $p.v_j = v_j$ must hold at state q_1, which means that $v_j \notin L$. This is a contradiction because $v_j \in L$.
  - If node v_j is neither a minimal point nor a source of corruption propagation at q_1' (i.e., $MP.v_j \vee SP.v_j = false$ at q_1'), then either action S_2 or action SC_1 is executed at node v_j in the step between q_1' and q_1, which sets $p.v_j$ to $v_{k'}$ other than v_{j-1}. Thus $v_j \notin L$. And this is a contradiction.

- Therefore, $d.v_{i-1} \geq d.v_{i-2} \geq \ldots \geq d.v_{i+1} \geq d.v_i$ holds at state q_1. Thus $d.v_{i-1} \geq d.v_i$ at state q_1', which contradicts the fact that $d.v_{i-1} < d.v_i$ at state q_1'.

Therefore, the loop L must have been in the system at state q_0, which contradicts the fact that there is no loop in the system at state q_0.

Thus, the claim holds.

□

[14] The length of a loop is the number of nodes in the loop.

[15] The addition as well as subtraction operation in this proof is on the modulo of l.

From Theorem 4, we can see that there is no loop in the system during stabilization if the possible faults in a system consist of only node fail-stop. This is because no loop will be formed just by the faults of fail-stop, which means that there is no loop in the system at any initial state of a system computation.

Theorem 5 (1-step Loop Breakage) *If there exists any loop in the system at an arbitrary state q_0, there will exist no loop in the system after one step of system computation starting at q_0.*

Proof: We prove this claim in two steps.

(I) First, for any loop L in the system at state q_0, it breaks within one step of system computation.

Let us assume that the length of loop L is l. Then we denote the set of nodes in the loop as $v_0, v_1, \ldots, v_{l-1}$ with $p.v_i = v_{i-1}$ $(i = 0, 1, \ldots, l-1)$[15].

If $d.v_0 = d.v_1 = \ldots = d.v_{l-1}$ holds at q_0, there are two possible cases for every node v_i in L. If v_i is a minimal point or a source of corruption propagation at q_0 (i.e., $MP.v_i \vee SP.v_i = true$ at q_0), then either action S_1 or action C_1 will be executed within one step of computation since state q_0, which sets $p.v_i$ to v_i itself. Thus loop L breaks. If v_i is neither a minimal point nor a source of corruption propagation at q_0 (i.e., $MP.v_i \vee SP.v_i = false$ at q_0), then either action S_2 or action SC_1 is executed at node v_i within one step of computation since q_0, which sets $p.v_i$ to v_j other than v_{i-1}. Thus loop L breaks too.

If $d.v_0 = d.v_1 = \ldots = d.v_{l-1}$ does not hold at q_0, there must exits node v_i $(0 \leq i \leq l-1)$ such that $d.v_{i-1} > d.v_i$. Then loop L will break at node v_i, according to the analysis in the proof for Theorem 4.

(II) Second, there is no new loop formed in any step of system computation. This trivially follows from the proof for Theorem 4.

Since every loop in the system at q_0 breaks in a step of system computation, and there is no new loop formed in any step of computation, there will be no loop in the system after a step of system computation starting at state q_0.

□

Corollary 2 (Fast Loop Removal) *Starting at an arbitrary state, there will be no loop in the system after one step of system computation.*

Proof: This claim trivially follows from Theorem 5.

□

5 Impact of network topology on local containment and stabilization

For the problem of minimum distance routing, the network topology of a system can affect the perturbation size as well as the range of contamination during stabilization upon perturbations in the system.

Observation 1 (Impact of network topology on local containment and stabilization) *Given two systems $G(V, E)$ and $G'(V', E')$, if the same faults occur in G and G' when they are at the same state q_0 where $V = V'$ and $E \subseteq E'$, and both systems reach the same state q_1 after the faults, then $P(G, q_1) \geq P(G', q_1)$ and $R_c(G, q_1) \geq R_c(G', q_1)$.*

The intuition behind the above observation is that, given a system $G(V, E)$, if we add some more edges to G and result in another system G' with "denser" edges, then the number of different shortest paths from every node in G' will be no less than that in G. Thus, if the same node i fail-stops in both G and G', the number of nodes that are perturbed due to the fail-stop of i in system G' will be no more than that in G. Therefore, the perturbation size in G' is no larger than that in G, if the same faults occur in both G' and G. Even if the perturbation size in G' and G is the same, the potential "mistakenly" initiated containment wave will propagate no farther in G' than in G as shown in Figure 4(b). Therefore, the range of contamination during stabilization in G' is no larger than that in G.

One example is shown in Figure 7. Figure 7(a) represents a system G in a legitimate state, Figure 7(b) represents a

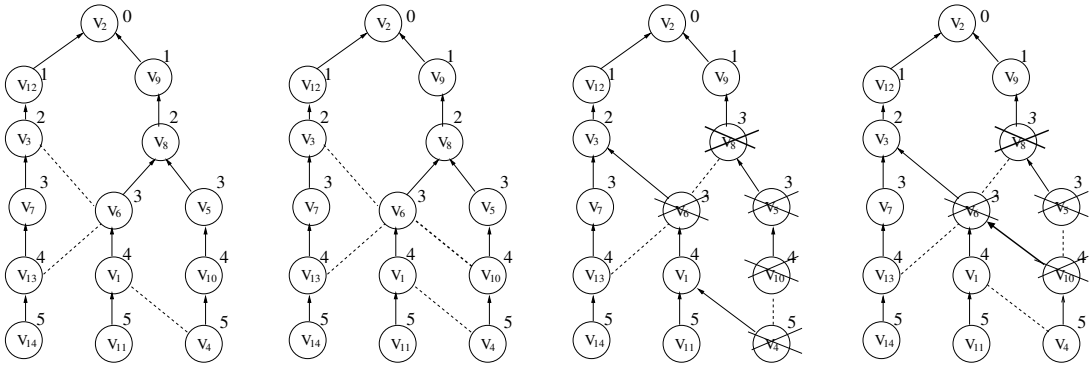


Figure 7: An example where "denser" edges can reduce the perturbation size and range of contamination during stabilization upon perturbations in a system

system G' in a legitimate state, and the only difference between G and G' is that edge $(v_6,\ v_{10})$ is not in G but in G'. If node v_8 fail-stops in both G and G', then the perturbation size in G is 4 (i.e., nodes v_6, v_4, v_{10}, and v_5 have to change their next-hop), whereas the perturbation size in G' is 3 (i.e., nodes v_6, v_{10}, and v_5 have to change their next-hop, but v_4 does not need to change its next-hop). If the state of v_8 is corrupted in both G and G' such that $d.v_8 = 3$, then the perturbed state of G and G' during stabilization can be that as shown in Figure 7(c) and (d) respectively, by which we can see that the range of contamination in G can be 3, whereas the range of contamination in G' can only be 2 at largest. The reason for this is that action SC_1 is enabled at v_{10} in G' and the "mistakenly" initiated containment wave from v_5 will not propagate beyond v_{10}.

In wireless networks, especially in wireless sensor networks [17], the edges tend to be dense because of dense node distribution and wireless transmission property (i.e., nodes within transmission range of one another are connected with one another). And this fact provides good chances for algorithms in those systems to contain faults more tightly in

terms of the constant factor in the asymptotic expression for the stabilization time and range of contamination during stabilization.

6 Related work

In [2], a self-stabilizing algorithm has been proposed for maintaining a spanning tree in a network, but the algorithm does not contain faults locally. In [3] and [1], time-optimal self-stabilizing algorithms for maintaining spanning trees have been proposed, but neither of them contains faults locally.

In [17], we proposed a local-containing algorithm for clustering as well as minimum distance routing in wireless networks, but the approach there is different from that in this paper because it takes advantage of such model properties as node density and geometric information in wireless networks. In [7], a loop-free distance-vector based routing algorithm was proposed, but it did not guarantee local containment of faults in a network. In [9] and [16], algorithms have been proposed to contain a single state-corruption during stabilization of a spanning tree, but these algorithms can deal with neither multiple faults nor fail-stop of nodes in a system.

In [8], a fault-containing self-stabilization algorithm for a consensus problem has been proposed, but it only considers the case where the topology is a linear array, and the range of contamination can be exponential in the perturbation size in the algorithm there. Moreover, the algorithm does not apply to the problem of minimum distance routing. In [12], an algorithm is proposed for locally mending faults, but it is not local in space because the representation of a solution is replicated across the whole system, and local containment is not guaranteed.

In [16] and [14], the concept of fault containment has been proposed. However, in [16], it only guarantees fault containment for the major part of system states, which is not strict enough to guarantee that the amount of work (for example, the number of actions executed) needed for a system to stabilize is a function of the perturbation size; moreover, it does not have the concept of range of contamination. In [14], it requires the range of contamination to be constant[16], which is so strict that it is not clear whether the concept applies to such problems as routing, where the locality of problems[17] is not constant. Our concept of local containment and stabilization concentrates on the amount of work needed during stabilization, and it is general enough to be applied to such problems as routing.

7 Conclusion

In this paper, we have designed an algorithm to achieve local containment and stabilization in minimum distance routing. One major machanism in our algorithm is to layer the diffusing computation in minimum distance routing into three waves: stabilization wave that guarantees the eventual convergence of a system to a legitimate state, containment wave that prevents a "mistakenly" initiated stabilization wave from propagating far away from where faults have occurred, and super-containment wave that prevents a "mistakenly" initiated containment wave from propagating far away. The interesting point behind the concept is that a node that has "mistakenly" initiated a diffusing wave w_1 (for example, stabilization wave or containment wave) will detect this within time proportional to the perturbation size,

[16] It is a special case of $\mathcal{F}$-local containment where the function $\mathcal{F}$ is a constant function.

[17] The locality of a problem is the maximum minimum distance between any two nodes that have to be involved in the definition of the problem.

and will initiate another diffusing wave w_2 (for example, containment wave or super-containment wave) that will catch up with and stop w_1 before it propagates far away. Moreover, there is no potential fault in super-containment wave, thus we do not need another diffusing wave to contain "possible" faults in super-containment wave, which enables the concept of layering. Another mechanism we have adopted to enable fast stabilization (i.e., stabilization time being proportional to perturbation size) is to avoid forming cycles during stabilization that can delay the convergence of a system, and to remove cycles in a system quickly if there is any at the initial state. We believe that these algorithmic concepts are not only suitable for achieving local containment and stabilization in minimum distance routing, they are also applicable to a richer class of distributed computing problems with the flavor of diffusing computation.

In order to characterize formally the properties of local containment and local stabilization in our algorithm, we have proposed such concepts as perturbation size, range of contamination, $\mathcal{F}$-local containing, and $\mathcal{F}$-local stabilizing. These concepts are not confined to the problem of minimum distance routing, and are applicable to any distributed computing problems. Thus, they are of their own interest, besides our algorithm for local containment and stabilization in minimum distance routing.

In our algorithm, we introduce delay in action execution to control the propagation speed of diffusing waves, but this will slow down the convergence of a system, because the stabilization time is only a linear function of the perturbation size instead of the system size. This is especially desirable in large-scale systems where faults generally occur only at a small part of the system, and it improves the availability, stability, and scalability of the system too.

For the problem of minimum distance routing, we have observed that denser edges in a system can potentially reduce the perturbation size and the range of contamination during stabilization upon perturbations in a system. This leads to an interesting question for network design as well as self-configuration algorithm design in (wireless) networks. That is, how to design or self-configure a system such that the perturbation size as well the range of contamination during stabilization is minimized.

In this paper, we assume that the weight of each edge is 1 for simplicity of presentation, but our algorithm can be applied with minor modification to the scenario where the weights are arbitrary non-negative values. This will be incorporated into our future work where we will study the application of our algorithm to the existing distance-vector based routing algorithms, such as RIP, BGP, DSDV, and AODV. Moreover, we will study how our algorithm can improve the stability of existing routing algorithms in the future work. And we will study in more detail the impact of network topology on local containment and stabilization in minimum distance routing.

References

[1] Sudhanshu Aggarwal and Shay Kutten. Time optimal self-stabilizing spanning tree algorithms. In *FSTTCS 1993*.

[2] Anish Arora and Mohamed G. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43(9):1026–1038, 1994.

[3] Yishay Mansour Boaz Patt-Shamir Baruch Awerbuch, Shay Kutten and George Varghese. Time optimal self-stabilizing synchronization. In *STOC 1993*.

[4] G. Robert Malan Craig Labovitz and Farnam Jahanian. Origins of internet routing instability. In *INFOCOM 1999*.

[5] Roger Wattenhofer Craig Labovitz, Ahba Ahuja and Srinivasan Venkatachary. The impact of internet policy and topology on delayed routing convergence. In *INFOCOM 2001*.

[6] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of The ACM*, 11:643–644, 1974.

[7] J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Transactions on Networking*, 1(1):130–141, 1993.

[8] Sukumar Ghosh and Xin He. Scalable self-stabilization. In *ICDCS'WSS 1999*.

[9] Sukumar Ghosh and Xin He. Fault-containing self-stabilization using priority scheduling. *Information Processing Letters*, 73:145–151, 2000.

[10] Christian Huitema. *Routing in the Internet*. Prentice-Hall, Inc., 1999.

[11] Mahesh Jayaram and George Varghese. Crash failures can drive protocols to arbitrary states. In *PODC 1996*.

[12] Shay Kutten and David Peleg. Fault-local distributed mending. In *PODC 1995*.

[13] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.

[14] Mikhail Nesterenko and Anish Arora. Local tolerance to unbounded byzantine faults. In *SRDS 2002*.

[15] Charles E. Perkins. *Ad Hoc Networking*. Addison Wesley, 2001.

[16] Ted Herman Sukumar Ghosh, Arobinda Gupta and Sriram V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *PODC 1996*.

[17] Hongwei Zhang and Anish Arora. GS^3: Scalable self-configuration and self-healing in wireless networks. In *PODC 2002*.