LSRP: Local Stabilization in Shortest Path Routing *

Hongwei Zhang Anish Arora †
Department of Computer and Information Science
The Ohio State University
2015 Neil Avenue, Dreese Lab 395
Columbus, OH 43210 USA

Abstract

In this paper, we formulate a notion of local stabilization, by which a system self-stabilizes in time proportional to the size of any perturbation that changes the network topology or the state of nodes. The notion implies that the part of the network involved in the stabilization includes at most the nodes whose distance from the perturbed nodes is proportional to the perturbation size. Also, we present LSRP, a protocol for local stabilization in shortest path routing. LSRP achieves local stabilization via two techniques. First, it layers system computation into three diffusing 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 the stabilization wave, the super-containment wave contains potential faults in the containment wave, and there is no potential fault in the super-containment wave. Second, LSRP avoids forming loops during stabilization, and it removes all transient loops within one round of computation. To the best of our knowledge, LSRP is the first protocol that achieves local stabilization in shortest path routing.

Keywords: Scalable and dependable shortest path routing, local stabilization, perturbation size, range of contamination, containment region

*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. This paper has been cleared through author affiliations.

†Email: {zhangho, anish}@cis.ohio-state.edu; Tel: +1-614-292-{1932, 1836}; Fax: +1-614-292-2911; Web: http://www.cis.ohio-state.edu/{~zhangho, ~anish}.

1 Introduction

A well-known ideal in networking is the ability to withstand failure or compromise of one or more regions in a network without impacting a large part of the network. Yet, in many instances, we find that even a small fault-perturbed region impacts a large part of the network, as the effects of the faults propagate to and contaminate far away nodes. An example is inter-domain routing in the Internet by the Border Gateway Protocol (BGP), where faults at some edge routers can propagate across the whole Internet [12, 18].

Unbounded fault propagation decreases not only the availability of a network but also its stability and scalability. Therefore, in large-scale networks such as the Internet and the emerging wireless sensor networks [14, 18, 19], it is desirable that faults be contained locally around the regions where they have occurred, and the time taken for a system to stabilize be a function $\mathcal{F}$ of the size of the fault-perturbed regions instead of the size of the system. We call this property *$\mathcal{F}$-local stabilization*.

Local stabilization in routing One problem where $\mathcal{F}$-local stabilization is critical but remains unsolved is the basic problem of shortest path routing in networks. Generally speaking, there are two categories of routing protocols: link-state and distance-vector. In link-state protocols, each node maintains the topological information of a whole network, and $\mathcal{F}$-local stabilization is impossible, since every single change in the network topology has to be propagated to every node in the network. In distance-vector protocols, each node only maintains the distance of and the next-hop on its shortest path to each destination in the network. Thus, $\mathcal{F}$-local stabilization is conceivable in distance-vector based routing protocols.

Distance-vector (and its variant, path-vector) protocols for the Internet, such as Routing Information Protocol (RIP) and BGP, have been long studied [8]. Distance-vector protocols for mobile ad hoc networks, such as Destination-Sequenced Distance-Vector (DSDV) and Ad hoc On-Demand Distance-Vector (AODV), have also been proposed [15]. In designing these protocols, researchers have typically concentrated on how to avoid routing loops and the count-to-infinity problem. Local stabilization is not guaranteed: small-scale local perturbations (such as memory overflow) can propagate globally across a whole network, due to the diffusing nature of these protocols [18], and result in severe instability [11, 18]. Moreover, the fault model has been typically limited to node and link faults such as crash, repair, and congestion; state corruption is not considered. However, several kinds of state corruption do arise as a result of misconfiguration and faulty software, and are known to be major causes for routing instability [12, 17, 18]. And theoretically speaking, even simple faults such as node crash and message loss, can drive a network into arbitrary states [9]. Therefore, $\mathcal{F}$-local stabilization is desirable, and not only in the presence of node/link crash, repair, and congestion but also in the presence of state corruption.

Contributions of the paper In this paper, we formulate the concepts of perturbation size, $\mathcal{F}$-local stabilization, and range of contamination, in order to characterize the local stabilization properties of systems. These concepts are generically applicable to distributed computing and networking problems. The perturbation size characterizes the extent of the perturbations at a system state. A system is $\mathcal{F}$-local stabilizing if and only if, starting at an arbitrarily perturbed state, every system computation reaches a legitimate state within time that is a function $\mathcal{F}$ of the perturbation size at the starting state. The range of contamination characterizes the maximum distance that perturbations in a system can propagate outward from the perturbed regions.

We also design LSRP (for Local Stabilizing shortest path Routing Protocol). Upon starting at an arbitrary state where the perturbation size is p, LSRP stabilizes to yield shortest path routes within $O(p)$ time, and the nodes affected by the perturbation are within $O(p)$ distance from the perturbed regions. Given two (or more) perturbed regions, LSRP stabilizes each region independently of and concurrently with the other(s) if the (half) distance between the regions is $\Omega(p')$, where p' is the size of the largest perturbed region. Moreover, LSRP guarantees local stabilization even if faults keep occurring in the perturbed regions with certain frequencies or for certain periods of time. To achieve local stabilization, LSRP exploits two techniques. One is to layer system computation into three diffusing 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. Each of these waves is confined locally around the perturbed regions. The stabilization wave corrects the state of 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 is no potential fault in the 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. The second technique is to avoid forming loops during stabilization. LSRP guarantees loop freedom during stabilization, and it removes all the transient loops, if there are any at the initial state, within at most one round of computation. Moreover, LSRP is scalable in the sense that each node only communicates with and maintains information about its 1-hop neighbors. To the best of our knowledge, LSRP is the first protocol that achieves local stabilization in shortest path routing.

We also discuss the impact of network topology on local stabilization in LSRP. We observe that higher edge density is beneficial in the sense that it can reduce the perturbation size, the range of contamination, and the stabilization time.

Organization of the paper In Section 2, we present the system, fault, and computation model. In Section 3, we formally define local stabilization, and analyze the properties of local-stabilizing systems. We present our LSRP protocol that solves the problem of local stabilization in shortest path routing in Section 4, and analyze its properties in Section 5. In Section 6, we discuss the impact of network topology in LSRP. Finally, we discuss related work is discussed in Section 7 and make concluding remarks in Section 8. In the Appendix, we discuss implementation issues related to LSRP, and give an example where higher edge density helps to contain faults more tightly.

2 Preliminaries

In this section, we present the system model, protocol notation, fault model, and computation model adopted in our work.

System model A system G is a connected undirected graph (V, E), where V and E are the set of nodes and the set of edges in the system respectively. Each node in the system has a unique ID. If nodes i and j can communicate with each other directly, then edge (i, j) is in E. For simplicity of presentation, the weight of each edge is assumed to be 1.

There is a clock at each node, and the ratio of clock rates between any two neighboring nodes in the system is bounded from above by α.

Two neighboring nodes in the system communicate with each other through shared memory[1]. A node

[1]We choose a shared memory model only for simplicity of presentation. Our protocol LSRP is readily adaptable to the message passing model (see Section 4.6).

can read its own variables as well as those of its neighbors, and it can write its own variables but not those of its neighbors. A node can also read variables of all its neighbors simultaneously.

Protocol notation and semantics We write protocols using a variant of the guarded command notation [2]. At each node, the protocol consists of a finite set of variables and actions. Each action consists of three parts: guard, statement, and range of execution delay. For convenience, we associate a unique name with each action. Thus, an action has the following form:

$$\langle name \rangle :: \langle guard \rangle \xrightarrow{[l,u]} \langle statement \rangle$$

The guard is a boolean expression over the protocol variables of the node and possibly those of its neighbors, the statement updates zero or more protocol variables of the node, l and u are the lower and upper bound on the execution delay of the action respectively, and $0 \leq l \leq u$. If $l = u = 0$, we write the action in the following form:

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

For an action named a, the lower and upper bound on its execution delay are denoted as $l.a$ and $u.a$ respectively. An action is enabled at time t if its guard evaluates to true at t. Then, an action a is executed at a node i at time t only if there exists e such that $e \in [l.a, u.a]$, a is continuously enabled from time $t - e$ to t, and either the value of the guard for a changes from false to true at $t - e$ or node i is executing another action at $t - e$. To execute an action, its statement is executed atomically.

Fault model A node or an edge is *up* if it functions correctly, and it is *down* if it fail-stops. In a system, nodes and edges that are up can fail-stop, nodes and edges that are down can become up and join the system, and the state of a node, i.e., the values of all the variables of the node, can be corrupted.

A fault occurs atomically, and it cannot overlap with the execution of actions and vice versa. The protocol actions of a node cannot be corrupted.

Computation model The topology of a system G is the subgraph $G'(V', E')$ of $G(V, E)$ such that $V' = \{i : i \in V \wedge\ i \text{ is up}\}$ and $E' = \{(i, j) : i \in V' \wedge j \in V' \wedge (i, j) \in E \wedge (i, j) \text{ is up}\}$. Due to faults, the system topology $G'(V', E')$ may change in the sense that the set of up nodes V' or the set of up edges E' changes over time. For example, node i is removed from V' when node i fail-stops. To deal with changes in system topology, we regard the state of G as the union of the current system topology and the states of all the nodes in this topology. At a system state q, the system topology and the state of a node i in this topology are denoted as $G.q(V.q, E.q)$ and $q(i)$ respectively. Given a system topology $G.q(V.q, E.q)$ and a problem specification, there exist a set of legitimate system states, denoted as $Q_l(G.q)$.

A system computation β is either a finite sequence $q_0, (a_1, t_1), q_1, (a_2, t_2), \ldots, q_n$, or an infinite sequence $q_0, (a_1, t_1), q_1, (a_2, t_2), \ldots, q_{r-1}, (a_r, t_r), q_r, \ldots$, of alternating system states (i.e. $q_0, q_1, \ldots$) and protocol actions (i.e. $a_1, a_2, \ldots$), where each state transition $q_{k-1}, (a_k, t_k), q_k$ ($k \geq 1$) means that the execution of action a_k at some node at time t_k changes the system state from q_{k-1} to q_k, and the following condition holds: for any two pairs (a_k, t_k) and $(a_{k'}, t_{k'})$ in β ($k \neq k'$), if a_k and $a_{k'}$ are executed at the same node, then $t_k \neq t_{k'}$ (i.e., at most one action can be executed at a node at any time). β is a finite sequence only if it ends with a state q_n, and there is no enabled action at q_n. A subsequence γ of β is called a computation segment if γ starts and ends with a state.

A system computation β can also be regarded as a sequence of *round*s. A round that starts at a state q_k ($k \geq 0$) is a minimal computation segment γ that starts at state q_k and every node that has an action a continuously enabled from time t_k to $t_k + u.a$ executes at least one action in γ (we assume t_0 is the time when β starts).

3 Local stabilization: concepts and properties

In this section, we first define concepts related to local stabilization, which are generic for distributed computing and networking problems, and then we present some notable properties of $\mathcal{F}$-local stabilizing systems.

3.1 Concepts related to local stabilization

In a distributed system, the variables that each node needs to maintain depend both on the problem and the protocol being used; some are inherent in the problem itself and independent of the protocol being used, while others are dependent on the protocol. For example, in the problem of shortest path routing, each node has to maintain the next-hop on its chosen shortest path to each destination node in order to be able to route a message to the destination, and the variable used to record this next-hop is inherent in the problem of shortest path routing. We call such variables that are inherent in the problem *problem-specific variables*. At a system state q, the values of the problem-specific variables of a node i is denoted as $q(i_p)$.

A node may be dependent on another node or edge in a distributed system, because, when faults occur at the latter, the former may have to change the values of its problem-specific variables in order for the system to stabilize to a legitimate state, no matter which protocol is used. For example, in the problem of shortest path routing, every node whose only shortest path to a destination goes through a node i or an edge e is dependent on i or e because it would have to change the next-hop on its shortest path to the destination if i or e fail-stopped.

Toward specifying the set of nodes that are dependent on a set of nodes V' and a set of edges E' at a legitimate system state q, we define the *dependent set of* V' *and* E'*at state* q, denoted by $DS_q(V', E')$, as:

$$\begin{cases} \{k : k \in V.q \wedge (\forall q' : q' \in Q_l(G_-) \Rightarrow q'(k_p) \neq q(k_p))\} & \text{if } V' \subseteq V.q \text{ and } E' \subseteq E.q \\ \{k : k \in V.q \wedge (\forall q' : q' \in Q_l(G_+) \Rightarrow q'(k_p) \neq q(k_p))\} \cup V' & \text{if } V' \cap V.q = E' \cap E.q = \emptyset \end{cases}$$

where $G_- = (V.q \setminus V', E.q \setminus E')$, $G_+ = (V.q \cup V', E.q \cup E')$. [2]

Now, we are ready to define the perturbation size at a system state, which is problem-specific and protocol independent. Intuitively, the perturbation size at a state q equals to the minimum number of nodes in $G.q$ whose states either have been corrupted by some transient faults or the values of whose problem-specific variables have to be changed in order for the system to stabilize to a legitimate state. It denotes the minimum amount of work needed to correct a perturbation that occurs in a system. Formally, perturbation size is defined as follows.

Definition 1 (Perturbation size) *The perturbation size at a system state q, denoted as* $P(q)$*, is*

$$\min_{q' \in Q_l} |A_{q'} \cup B_{q'}|$$

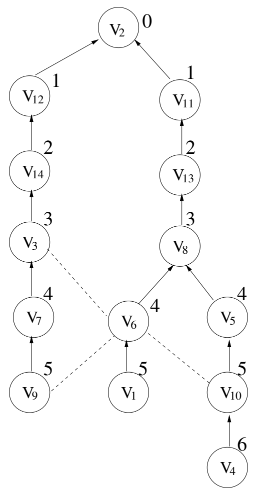
[2]The node set V' and edge set E' should be such that $G_-(V_-, E_-)$ and $G_+(V_+, E_+)$ are valid graphs. That is, for every edge $e = (i, j)$ in the edge set E', $i \in (V.q \cup V')$ and $j \in (V.q \cup V')$; and for every node i in the node set V', $\{(i, j) : j \in (V.q \cup V') \wedge (i, j) \in (E.q \cup E')\} \subseteq E'$.

where

Q_l *is the set of all possible legitimate system states*,
$A_{q'} = \{i : i \in (V.q \cap V.q') \wedge q(i) \neq q'(i)\}$,
$B_{q'} = \{i : i \in V.q \wedge i \in (DS_{q'}(V.q' \setminus V.q, E.q' \setminus E.q) \cup DS_{q'}(V.q \setminus V.q', E.q \setminus E.q'))\}$

Then, the *set of potentially perturbed node set at state* q, denoted as $PP(q)$, are defined as $\{A_{q'} \cup B_{q'} : q' \in Q_l \wedge |A_{q'} \cup B_{q'}| = P(q)\}$, where Q_l, $A_{q'}$, and $B_{q'}$ are the same as in Definition 1.

For example, in the problem of shortest path routing, Figure 1 represents a legitimate system state. When



In the figure, each circle represents a node in the system, the string in a circle represents the ID of the node, node v_2 is a destination node, and the number besides each circle represents the distance from that node to v_2. A directed edge $< v_i, v_j >$ means that v_j is the next-hop on the chosen shortest path from v_i to v_2, and an undirected dashed edge (v_i, v_j) means that v_i and v_j are neighbors in the system, but neither is v_j on the chosen shortest path from v_i to v_2, nor is v_i on the chosen shortest path from v_j to v_2.

Figure 1: Example of perturbation size in a system when faults occur

the system is at this state, if state corruption occurs at node v_8, then the perturbation size at the state after the corruption is 1 and the set of potentially perturbed nodes is $\{v_8\}$, since only v_8 needs to change its state in order for the system to stabilize to legitimate state, and at least one node in the system needs to change its state in order for the system to stabilize; however, if node v_8 fail-stops, then the perturbation size is 3 and the set of potentially perturbed nodes is $\{v_6, v_5, v_{10}\}$, since nodes v_6, v_5, and v_{10} have to change their next-hop on their shortest paths to v_2, while all the other nodes in the system don't need to.

In contrast to the concept of perturbation size which is protocol independent, the concept of $\mathcal{F}$-local stabilization is protocol-dependent.

Definition 2 ($\mathcal{F}$-local stabilization) *A system G is $\mathcal{F}$-local stabilizing if and only if*

> *Starting at an arbitrary state q, every computation of G reaches a legitimate state within $\mathcal{F}(P(q))$ time, where $\mathcal{F}$ is a function and $P(q)$ is the perturbation size at state q.*

If a system is $\mathcal{F}$-local stabilizing, we say that $\mathcal{F}$-local stabilization is guaranteed in the system. If $\mathcal{F}$ is a linear function, we say that the system is linear-local stabilizing (or local stabilizing, for simplicity).

Given an $\mathcal{F}$-local stabilizing system and a system computation β that starts at state q and reaches a legitimate state q' , the perturbed node set at q, denoted as $PN(q)$, is defined such that $PN(q) \in PP(q)$ and $|PN(q)| = max_{\mathrm{S} \in PP(q)} |\mathrm{S} \cap \{i : i \in G.q \wedge q(i) \neq q'(i)\}|$. A node i is said to be *perturbed* at q if $i \in PN(q)$, otherwise, it is said to be *healthy* at q. Then, the range of contamination, denoted by $R_c(q)$, is defined as the the maximum minimum distance between the set of perturbed nodes at q and the set of nodes that are healthy at q but contaminated during stabilization. That is,

$$R_c(q) \ = \ \max_{i \in S_c} dist(i, S_p)$$

where

$S_c = \{i : i \in V.q \wedge i$ is healthy at $q \wedge$ some protocol action is executed at i before G stabilizes$\}$,
$S_p = \{j : j \in V.q \wedge j$ is perturbed at $q\}$,
$dist(i, S_p) = \min_{j \in S_p} dist(i, j, G.q)$,
$dist(i, j, G.q) =$ the number of hops on the shortest path between i and j in $G.q$.

3.2 Properties of $\mathcal{F}$-local stabilizing systems

A set of nodes S are *contiguous* at a system state q if $S \subseteq V.q$ and the subgraph of $G.q(V.q, E.q)$ on S is connected, i.e., the graph $G'(V', E')$ is connected, where $V' = S$ and $E' = \{(i, j) : i \in S \wedge j \in S \wedge (i, j) \in E.q\}$. A maximal set of perturbed nodes that are contiguous is called a *perturbed region*. Then the following properties hold for a $\mathcal{F}$-local stabilizing system G:

- Starting at an arbitrary state q, the maximum distance that faults can propagate outward from the perturbed regions is $O(\mathcal{F}(P(q)))$, i.e., the range of contamination is $O(\mathcal{F}(P(q)))$. Therefore, every node that is $\Omega(\mathcal{F}(P(q)))$ hops away from the perturbed regions at state q will not be contaminated by the perturbation. This property is presented in Proposition 1.

- Starting at an arbitrary state q where the perturbed regions are $\Omega(\mathcal{F}(P(q)))$ hops away from one another, the stabilization of one perturbed region is independent of and concurrent with that of the other perturbed regions, and the time taken for the system to stabilize only depends on the size of the largest perturbed region. However, if the perturbed regions at q are close to one another, the stabilization of one perturbed region may delay that of the other perturbed regions.These properties are presented in Proposition 2, Corollary 1, and Proposition 3.

- Because it is $\mathcal{F}$-local stabilizing, the availability of G is high in the sense that it stabilizes quickly after perturbations and the impact of perturbations is contained locally around where they occur. Moreover, G can tolerate high frequency of repeated faults while still guaranteeing local fault containment. This property is presented in Proposition 4.

Formally, these properties are presented as the following propositions.

Proposition 1 (Local fault containment) *For a $\mathcal{F}$-local stabilizing system G, starting at an arbitrary state q, the range of contamination during stabilization is $O(\mathcal{F}(P(q)))$.*

When the set of perturbed nodes are not contiguous at a system state q, there are multiple perturbed regions $S_0, S_1, \ldots, S_m$ ($m \geq 1$) in the system. For each perturbed region S_i ($i = 0, \ldots, m$), its *containment region*, denoted as CR_i, is the union of S_i and the set of nodes that are contaminated during stabilization because of the existence of S_i. Two containment regions CR_i and CR_j ($i \neq j$) are *disjoint* if there don't exist any two neighboring nodes k and k' such that $k \in CR_i$ and $k' \in CR_j$, otherwise, they are *adjoining*. Multiple containment regions $CR_0, CR_1, \ldots, CR_m$ ($m \geq 1$) are disjoint if there don't exist any two containment regions CR_i and CR_j ($i \neq j$) that are adjoining; they are said to be adjoining if, for any two containment regions CR_i and CR_j ($i \neq j$), either CR_i and CR_j are adjoining or there exist a sequence of containment region $CR_{k_0}, CR_{k_1}, \ldots, CR_{k_t}$ such that CR_i are adjoining with CR_{k_0}, CR_{k_n} are adjoining with $CR_{k_{n+1}}$ ($n = 0, \ldots, t-1$), and CR_{k_t} are adjoining with CR_j.

Proposition 2 (Local stabilization when containment regions are disjoint) *Starting at an arbitrary state q where the perturbed regions are $S_0, S_1, \ldots, S_m$ ($m \geq 1$) and their containment regions are disjoint, a $\mathcal{F}$-local stabilizing system self-stabilizes to a legitimate state within $\mathcal{F}(\max_{i\in 0..m}|S_i|)$ amount of time, and the range of contamination is $\mathcal{F}(\max_{i\in 0..m}|S_i|)$.*

Given any two perturbed regions S_i and S_j ($i \neq j$) at a state q, the half-distance between S_i and S_j is half of the minimum distance from a node in S_i to another node in S_j, that is, $\min_{k\in S_i, k'\in S_j}\lfloor\frac{dist(k,k',G.q)}{2}\rfloor$, where $dist(k, k', G.q)$ is the number of hops on the shortest path between node k and k' in graph $G.q$.

Corollary 1 *Starting at an arbitrary state q_0 where the perturbed regions are $S_0, S_1, \ldots, S_m$ ($m \geq 1$) and the half-distance between any two of them is $\Omega(\mathcal{F}(\max_{i\in 0..m}|S_i|))$, a $\mathcal{F}$-local stabilizing system self-stabilizes to a legitimate state within $\mathcal{F}(\max_{i\in 0..m}|S_i|)$ amount of time, and the range of contamination is $\mathcal{F}(\max_{i\in 0..m}|S_i|)$.*

Proposition 3 (Local stabilization when containment regions are adjoining) *Starting at an arbitrary state q where the perturbed regions are $S_0, S_1, \ldots, S_m$ ($m \geq 1$) and their containment regions are adjoining, a $\mathcal{F}$-local stabilizing system self-stabilizes to a legitimate state within $\mathcal{F}(\sum_{i=0}^{m}|S_i|)$ amount of time, and the range of contamination is $\mathcal{F}(\sum_{i=0}^{m}|S_i|)$.*[3]

Proposition 4 (Local fault containment in spite of persistent faults) *If some transient faults that drive a $\mathcal{F}$-local stabilizing system G to state q keeps occurring in the system, but the interval between any two consecutive occurrences of such faults is larger than $\mathcal{F}(P(q))$, then the range of contamination is still $O(\mathcal{F}(P(q)))$. That is, faults are contained around where they are occurring within distance $O(\mathcal{F}(P(q)))$.*

4 Protocol LSRP

In this section, we first specify the problem of local stabilization in shortest path routing. Then we explain the limitations of existing distance-vector based routing protocols, present the protocol concepts underlying LSRP, and finally present LSRP and its design.

4.1 Problem statement

The problem is to design a protocol that, given a system $G(V, E)$ and a destination node r in the system, constructs and maintains a spanning tree T (called *shortest path tree*) of G that meets the following requirements:

- Node r is the root of the shortest path tree T;
- $(\forall i : i \in V \Rightarrow dist(i, r, T) = dist(i, r, G))$, where $dist(i, r, T)$ and $dist(i, r, G)$ are the minimum distance between nodes i and r in T and G respectively; (that is, the path from every node i to r in T is a shortest path between i and r in G.)
- The system G is $\mathcal{F}-$local stabilizing.

[3] *This proposition comes from the definitions of perturbation size and $\mathcal{F}$-local stabilizing.*

4.2 Limitations of existing distance-vector based routing protocols

Existing distance-vector routing protocols are based on the distributed Bellman-Ford algorithm [13]. In these protocols [4][8], 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 i and $d.j = \min\{d.k : k \text{ is a neighbor of } i\}$, 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 protocols, faults cannot be contained around where they have occurred, and $\mathcal{F}$-local stabilization is not guaranteed, which results in routing instability.

One example is shown in Figure 2. For the same system in Figure 1, Figure 2(a) represents a system

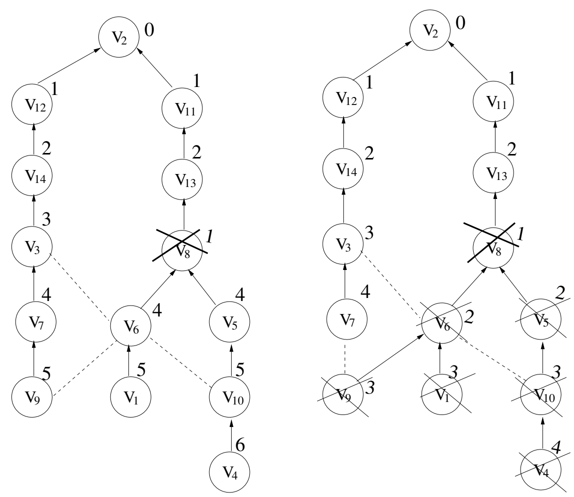


Figure 2: Example of fault propagation in existing distance-vector based routing protocols

state where the state of node v_8 is corrupted such that $d.v_8 = 1$. Ideally, v_8 should correct its state such that $d.v_8 = 3$, and all the other nodes in the system remain unaffected by the state corruption at v_8. However, in existing distance-vector based routing protocols, it is possible that nodes v_6 and v_5 detect the change of $d.v_8$ before v_8 corrects its state. Then both v_6 and v_5 will change their state correspondingly such that $d.v_6 = d.v_5 = d.v_8 + 1 = 2$. And the same happens at nodes v_9, v_1, v_{10}, and v_4. Therefore, the fault at v_8 propagates to nodes v_6, v_5, etc., and the perturbed system state after the fault propagation from v_8 is shown in Figure 2(b). Even though the system will stabilize to a legitimate state later, nodes far away from v_8, such as v_4 and v_9, have been contaminated by the state corruption at v_8, and the time taken for the system to stabilize depends on the diameter of the system instead of the perturbation size. Furthermore, node v_9 has changed its route to destination v_2 because of the fault propagation, which leads to route flapping, a severe kind of routing instability.

In a large-scale distributed system, there can be multiple faults happening to different nodes in the system, and the fault at any single node can propagate across and contaminate the whole system. This kind of unbounded fault propagation can result in severe routing instability [12] and reduce the availability as well as scalability of a system. Therefore, protocols that can contain faults locally around where they have occurred and let a system stabilize within time that is only a function of the perturbation size are strongly desired in these fault-prone systems [18].

4.3 Protocol concepts

In the example shown in Figure 2, the state corruption at v_8 can propagate far away until it reaches the leaves of the shortest path tree, and the time taken for the system to stabilize depends on its diameter instead of the perturbation size. The reasons for the unbounded fault propagation and slow stabilization are 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, v_6 as well as v_5 detects that $d.v_8$ decreases. Because neither v_6 nor v_5 knows that the new state of v_8 is a corrupted one, both v_6 and v_5 update their state according to the corrupted state of node v_8, and the state corruption 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 3). Then, its neighbors v_6 and v_5 correct their corrupted states, 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 until it reaches the leaves of the shortest path tree, hence the time taken for the system to stabilize depends on the system diameter instead of the perturbation size at the initial state.

In short, the reason why faults propagate and local stabilization is violated is that the "correction" action always lags behind the "fault propagation" action. Therefore, one approach to contain faults locally and achieve local stabilization is to guarantee that the node that is the source of fault propagation (for example, node v_8) will detect the existence of fault propagation, and initiate a "containment" action that can catch up with and stop the "fault propagation" action before faults propagate far away. We develop this approach as follows.

Layering of diffusing waves In shortest path routing, the system computation can be regarded as a diffusing computation[4]. To achieve local stabilization in shortest path routing, we layer the diffusing computation into three diffusing waves: the *stabilization wave*, the *containment wave*, and the *super-containment wave* (see Figure 3). Each wave has a different propagation speed, with the stabilization wave propagated at the lowest speed, containment wave at intermediate speed, and super-containment wave at the highest speed.

The stabilization wave is a diffusing computation that implements the basic distributed Bellman-Ford algorithm with some changes in order to cooperate with the containment wave. A stabilization wave can

[4]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 Bellman-Ford based routing protocols, 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 reaches i.

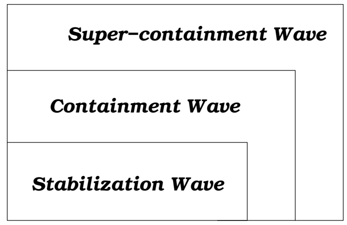


Figure 3: Layering of diffusing waves in shortest path routing

propagate the "correction" action that makes a system converge to a legitimate state, but it can also propagate faults far away from where they initially occurred, as shown in Figure 2. Therefore, in order to prevent a stabilization wave from propagating faults unbounded, the containment wave is introduced.

Containing the stabilization wave A containment wave is initiated from a node that is a "potential source of fault propagation" (simply called source of fault propagation hereafter). A node is a source of fault propagation if its distance value is the smallest among nodes within its 1-hop neighborhood (including itself) that are not involved in any containment wave, but the node is neither the destination node nor is its distance value 0 even if it is the destination node. For example, node v_8 in Figure 2(a) is a source of fault propagation. The containment wave propagates along the same paths as those by the stabilization wave which propagates faults from the source of fault propagation. And the containment wave propagates faster than the stabilization wave such that it is able to catch up with and stop the stabilization wave.

However, a containment wave can be mistakenly initiated due to state corruption. For example, in Figure 2(a), if the state of v_8 is corrupted such that $d.v_8 = 4$, node v_5 will become a source of fault propagation and a containment wave will be initiated at v_5. Therefore, in order to prevent a mistakenly initiated containment wave from propagating unbounded, the super-containment wave is introduced.

Fault tolerance of the containment wave A node that has mistakenly initiated a containment wave will detect that it should not have initiated the containment wave after time at most proportional to the perturbation size. Then it will initiate a super-containment wave that propagates along the same paths as those by the mistakenly initiated containment wave. Since the super-containment wave propagates faster than the containment wave, the super-containment wave will catch up with and stop the containment wave.

In order for the above wave-layering approach to work, there should exist no potential fault in the super-containment wave, otherwise, there is no end to the layering procedure. This is achieved by ensuring that the super-containment wave only uses variables defined for the stabilization wave and containment wave, and no extra variable is introduced for the super-containment wave.

Loop freedom In the basic distributed Bellman-Ford algorithm, loops can form during stabilization, which leads to the bouncing effect and count-to-infinity problem [8] that delay the stabilization of a system and violate the time constraint of local stabilization. Therefore, in order to circumvent these two problems, our protocol avoids forming loops during stabilization, which, together with local fault containment, guarantees that the stabilization time is a function of the perturbation size in the worst case. Interestingly, loops can be avoided during stabilization just via the containment wave. The intuition is that a node which can select one of its descendant as its new parent (i.e. its next-hop) in the basic distributed Bellman-Ford algorithm becomes a source of fault propagation according to our definition. Therefore, a containment wave will be initiated at such a node, which guarantees loop freedom because no loop is formed in any containment wave.

4.4 Protocol design

The protocol LSRP (Local Stabilizing shortest path Routing Protocol) is shown in Figure 4, where the constants, variables, and protocol actions for each node i in a system are presented.

Protocol $LSRP.i$

Constant r : node_id

$d_s,\ d_c,\ d_{sc},\ L,\ U$: real

Var $d.j$: integer

$p.j$: node_id

$ghost.j$: boolean

k : node_id

Parameter j : node_id

Actions

$\langle S_1 \rangle ::$ $MP.i \wedge p.i \neq i \longrightarrow p.i := i$

[]

$\langle S_2 \rangle ::$ $SW.i.j \wedge \neg ghost.j \xrightarrow{[d_s+L, d_s+U]} d.i, p.i := d.j + 1, j;$

$ghost.i := false$

[]

* * * * *

$\langle C_1 \rangle ::$ $\neg ghost.i \wedge (SP.i \vee CW.i) \xrightarrow{[d_c+L, d_c+U]} \underline{if}\ SP.i \ \rightarrow\ p.i := i\ \underline{fi}\ ;$

$ghost.i := true$

[]

$\langle C_2 \rangle ::$ $ghost.i \wedge \neg(\exists k : k \in N.i \wedge p.k = i \wedge d.k = d.i + 1) \longrightarrow$

$ghost.i := false;$

$\underline{if}\ i = r\ \rightarrow\ d.i, p.i := 0, i$

[]

$i \neq r \wedge PS.i.j\ \rightarrow\ d.i, p.i := d.j + 1, j$

[]

$i \neq r \wedge \neg(\exists k : PS.i.k)\ \rightarrow\ d.i, p.i := \infty, i$

$\underline{fi}$

[]

* * * * *

$\langle SC \rangle ::$ $ghost.i \wedge SCW.i \xrightarrow{[d_{sc}+L, d_{sc}+U]} ghost.i := false$

Figure 4: LSRP: local stabilization in shortest path routing

Constants There are six constants used in the protocol, that is, r, d_s, d_c, d_{sc}, L, and U. 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 the stabilization wave, containment wave, and super-containment wave respectively. L and U can be any constant numbers as long as $0 \leq L \leq U$. (When implementing LSRP in a real network, L and U can be chosen to reflect network properties, as discussed in Section 4.6.)

To guarantee local stabilization, d_s, d_c, and d_{sc} should be such that $d_s > \max\{\alpha \cdot d_c,\ d_c + U - L\}$, $d_c > \max\{\alpha \cdot d_{sc},\ d_{sc} + U - L\}$, and $d_{sc} \geq 0$, where α is the upper bound on the ratio of clock rates between any two neighboring nodes. [5]

Variables As in existing distance-vector based routing protocols, each node i maintains the two variables $d.i$ and $p.i$, where $d.i$ records the shortest distance from i to r, and $p.i$ records the next-hop on the shortest path from i to r (i.e., the parent of i in the shortest path tree rooted at r). However, in order to achieve local stabilization, each node i maintains another boolean variable $ghost.i$. $ghost.i$ 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 i. A dummy variable k is also used.

Protocol actions As mentioned in Section 4.3, the diffusing computation in LSRP consists of three diffusing waves: the stabilization wave, the containment wave, and the super-containment wave. Among the five actions in LSRP, S_1 and S_2 are for the stabilization wave, C_1 and C_2 for the containment wave, and SC for the super-containment wave.

- *Stabilization wave*

  The stabilization wave guarantees that a system eventually stabilizes to a legitimate state, and it implements the distributed Bellman-Ford algorithm with some changes to cooperate with containment wave.

  - $\underline{Action\ S_1}$: if node i is a minimal point (i.e., $MP.i = true$) but $p.i \neq i$, it sets $p.i$ to i.

    $MP.i$ is defined as

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

    That is, a node i is a minimal point if it is the destination node and $d.i = 0$, or if it has initiated a containment wave that has not finished.

  - $\underline{Action\ S_2}$: if node i should propagate a stabilization wave from node j (i.e., $SW.i.j = true$) that is not being involved in any containment wave, and this condition continuously held in the past e amount of time where $e \in [d_s + L, d_s + U]$, then i sets j as its parent, and sets $d.i$, $ghost.i$ to $d.j + 1$, $false$ respectively.

    $SW.i.j$ is defined as

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

    That is, node i should propagate a stabilization wave from node j if

    - node 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
    - if j is not the current parent of i (i.e., $j \neq p.i$), then the distance value of j is less than that of $p.i$ unless $p.i$ is not a neighbor of i or is involved in a containment wave; or if j is the current parent of i, then the distance value of i is not equal to that of j plus 1.

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

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

- *Containment wave*

  The containment wave prevents a stabilization wave from propagating faults far away from where they have occurred.

  - Action C_1: if node i is not being involved in any containment wave (i.e., $ghost.i = false$), but it is either a source of fault propagation (i.e., $SP.i = true$) or it should propagate a containment wave from its parent (i.e., $CW.i = true$), and this condition continuously held in the past e amount of time where $e \in [d_c + L, d_c + U]$, then i sets $ghost.i$ to $true$ in order to initiate or propagate a containment wave. Moreover, if i is a source of fault propagation, it sets $p.i$ to i.

    $SP.i$ is defined as

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

    That is, a node i is a source of fault propagation if its distance value is the smallest among nodes within its 1-hop neighborhood (including itself) that are not involved in any containment wave, but it is either not the destination node and its distance value is not consistent with that of its parent, or its distance value is not 0 even though it is the destination node.

    $CW.i$ is defined as

    $$p.i \in N.i \wedge ghost.(p.i) \wedge d.i = d.(p.i) + 1 \ \wedge \neg(\exists k : k \in N.i \wedge \neg ghost.k \wedge d.k < d.i)$$

    That is, a node i should propagate a containment wave from its parent $p.i$ if $p.i$ is a neighbor of i and is involved in a containment wave, i has copied the corrupted distance value of $p.i$ (i.e., $d.i = d.(p.i) + 1$), and i does not have a neighbor j that is not involved in any containment wave and whose distance value is less than that of i.

    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; moreover, when a node i propagates the containment wave from its parent by the execution of C_1, the pointer to its current parent (i.e., $p.i$) should be left unmodified.

  - Action C_2: if node i is involved in a containment wave, but i has no child k that is perturbed due to the state corruption at i (i.e., $p.k = i$ and $d.k = d.i + 1$), then i sets $ghost.i$ to $false$, and

    - if i is the destination node, then i sets $d.i$ and $p.i$ to 0 and i respectively;
    - if i is not the destination node, then i sets $d.i$ and $p.i$ to $d.j + 1$ and j respectively, if there exists a parent substitute j of i (i.e., $PS.i.j = true$); otherwise, i sets $d.i$ and $p.i$ to ∞ and i respectively.

      $PS.i.j$ is defined as

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

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

  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 at the initiator of the containment wave.

- *Super-containment wave*

  The super-containment wave prevents a mistakenly initiated containment wave from propagating unbounded, and corrects faults in the containment wave.

  – Action SC: if node i is 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., $SCW.i = true$), and this condition continuously held in the past e amount of time where $e \in [d_{sc} + L, d_{sc} + U]$, then 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 and $d.i = 0$; or

    ◦ it is not the destination node, and neither is it a source of fault propagation nor is its parent involved in any containment wave.

4.5 Example revisited

Now, let us revisit the example as shown in Figure 2 by examining how the system behaves if LSRP is adopted. Suppose a system is at a state as shown in Figure 2(a) where the state of v_8 is corrupted such that $d.v_8 = 1$. At this state, the guard for action C_1 at v_8 evaluates to true (because v_8 is a source of fault propagation, by definition), so does the guard for action S_2 at v_6 and v_5. However, since $d_s > \alpha \cdot d_c$, v_8 will execute C_1 before v_6 and v_5 get chance to execute S_2. After C_1 is executed at v_8, $ghost.v_8$ is set to true, which disables action S_2 at v_6 and v_5. Then, action C_2 will be executed at v_8, which corrects $d.v_8$ to 3, and leads the system to a legitimate state. In the above process, only action C_1 and C_2 are executed at v_8, and no action is executed elsewhere. Therefore, no other nodes in the system is affected by the state corruption at v_8, which is the ideal case achievable.

4.6 Implementation issues

Here, we briefly discuss how to implement LSRP in message passing communication model in real networks.

In Section 4.4, we mentioned that, in theory, the two protocol constants L and U used in LSRP can be arbitrary as long as $0 \leq L \leq U$. However, L and U affect the stabilization speed in the sense that they affect the constant factor in the asymptotic notation of the stabilization time: the larger the L and U, the slower the stabilization. Therefore, it is desirable that L and U not be too large, in order to guarantee fast stabilization. On the other hand, no matter which L and U are chosen, LSRP always guarantees local stabilization (as discussed in Section 5). Therefore, in order to simulate the shared memory communication model in real networks while implementing LSRP, we can choose L and U such that they reflect the communication delay in message exchange between neighboring nodes in a specific network. This is analyzed as follows.

In a shared memory communication model as discussed in Section 2, we assume that a node can read protocol variables of all its neighbors simultaneously. This assumption can be realized through an algorithm for dining philosophers that contains faults locally and guarantees that no two neighboring nodes execute actions simultaneously [14], by which a node can collect the state of all its neighbors by communicating with them via message passing. The time taken for a node to collect the state of all its neighbors via the algorithm can be reflected in L and U. That is, L and U can be chosen to be the lower and upper bound on the time taken for a node to collect the state of all its neighbors via message passing.

5 Protocol analysis

In this section, we present the fixpoint[6] of LSRP and the property of local stabilization in a system where LSRP is used. We also present the properties of loop freedom during stabilization and quick loop removal in LSRP, which are not only necessary for local stabilization in routing, but also critical for improving network resource utilization and quality of communication.

5.1 Property of local stabilization

Given a system topology $G'(V', E')$, let $\mathcal{L} \equiv (\forall i : i \in V' \Rightarrow \neg ghost.i \wedge LH.i)$, where $LH.i$ is defined as

$$\begin{array}{l}(i = r \Rightarrow d.i = 0 \wedge p.i = i)\wedge \\ (i \neq r \Rightarrow d.i = d.(p.i) + 1 \wedge p.i \in N.i \wedge (\forall k : k \in N.i \Rightarrow d.(p.i) \leq d.k))\end{array}$$

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

Moreover, $\mathcal{L}$ is a fixpoint (or stable state) of LSRP, and every system where LSRP is used self-stabilizes to a state in $\mathcal{L}$, upon starting at an arbitrary state.

Theorem 1 (Fixpoint) *$\mathcal{L}$ is a fixpoint of LSRP, i.e., any state in $\mathcal{L}$ is a stable state of the system where LSRP is used.*

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

Therefore, no action is enabled at any state in $\mathcal{L}$, and $\mathcal{L}$ is a fixpoint of the protocol.

□

Theorem 2 (Self-stabilization) *Starting at an arbitrary state, every computation of a system where LSRP is used is guaranteed to reach a state in $\mathcal{L}$.*

Proof: Let D be the diameter of system $G(V', E)$.

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

(i) Starting at an arbitrary state, the system is guaranteed to reach a state in $\mathcal{L}.0$.

[6]The fixpoint of a protocol is a system state where there is no enabled protocol action. Therefore, it is also a stable state of the system.

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

$\mathcal{L}.l$ is closed under system execution; that is, once $\mathcal{L}.l$ 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 $\mathcal{L}.l$, the system is guaranteed to reach a state in $\mathcal{L}.(l+1)$ within a constant amount of time.

(iv) $\mathcal{L}.D \equiv \mathcal{L}$

For our protocol, $\mathcal{L}.0, \mathcal{L}.1, \ldots, \mathcal{L}.D$ are defined as follows,

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

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

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

We prove the stabilization of $\mathcal{L}$ 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 $\mathcal{L}.0$ 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 $\mathcal{L}.0'$ in $O(N)$ amount of time, where $\mathcal{L}.0' \equiv (\forall i : i \in V' \Rightarrow \neg ghost.i \wedge \neg SP.i)$.

Since every system computation can be regarded as a sequence of rounds, we number the sequence of rounds as $s_0, s_1, s_2, \ldots$ and so on. s_0 is an *empty round* 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 round immediately prior to s_{i+1}, and s_{i+1} is the round immediately behind s_i. Because there can be at most one action being enabled at each node in protocol LSRP, and the time taken to execute an action is bounded from above by U, each round takes at most $d_s + U$ 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 round 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 round s_j, $action_k$ is executed at node k';
$s_j(x)$	:	the value of variable x at the end of round 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 round s_j ($j = 0, 1, \ldots$), $CT_i(s_j)$ and $WC_i(s_j)$ are defined inductively as follows:

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

and

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

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

Intuitively, if a node i is not a source of fault propagation at system state q_0, its containment tree CT_i is empty during any system computation. If node i is a source of fault 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 fault 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 round 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 round s_{j+1}, either an ancestor of k in s_j is in V_j or node k becomes a source of fault propagation at the end of s_j or in s_{j+1} 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 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 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 at least one in every round of system computation until reaching its maximum value, and then keeps decreasing by at least one in every round of computation until the tree becomes an empty tree. The reasons are as follows:

i. For any non-empty containment tree CT_i, either $Depth(CT_i(s_{j+1})) \geq Depth(CT_i(s_j)) + 1$ 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} ($j = 0, 1, \ldots$).
Whether the depth of the tree can increase in round 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 round s_j, action S_2 and C_2 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 action S_2 is not 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., $s_{j+1}(C_1, k') \wedge \neg(\exists k'' : (k'' = k \vee Anc(k'', k, CT_i(s_j))) \wedge s_{j+1}(S_2, k'')))$, node k' will be added to V_a in s_{j+1}. Moreover, it is possible for a child of k' to be added to V_a later in s_{j+1}. Therefore, $Depth(CT_i(s_{j+1})) \geq Depth(CT_i(s_j)) + 1$ holds in thise case.
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 $s_j(SP.l) = true$) or S_2 in round 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 can be enabled at node k in s_{j+1} too, but its execution cannot increase or retain the depth of the containment tree of node i from round 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 action S_2 is not 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 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 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 round s_j ($j = 1, 2, \ldots$), it keeps decreasing in every round 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 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 round 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 can also be enabled at node k in s_{j+1}, but its execution only removes node k from CT_i.

If the depth of the containment tree CT_i decreases in a round 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' is such that $s_{j-1}(SP.k') = true$) or S_2, 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 round s_{j+1} according to the analysis above. Thus, the depth of CT_i will decrease in round 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 rounds R, and $k' \in CT_{i'}$ in a set of rounds 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 rounds R, and $k' \in CT_{i'}$ in a set of rounds R', the above claim still holds because of the following facts:

- If there exists a round 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 that can set k' as the parent of k will not be executed at k in s_{j+1}, neither will action S_2 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 round s_j such that $s_j \in R$ and $s_j \in R'$, but exists a round 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 action S_2 that can set k' as a child of k or set k as a child of k' will not 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 round 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 $\mathcal{L}.0'$ in $O(N)$ time. If $\mathcal{L}.0'$ holds at q_0, the claim trivially holds. So, we only need to prove that the system converges from a state q_0 in $\neg\mathcal{L}.0'$ to a state in $\mathcal{L}.0'$ in $O(N)$ time.

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

$$
\begin{aligned}
f(s_j) \;&=\; \langle G(s_j), T(s_j)\rangle, \text{ where} \\
&\quad G(s_j) \;=\; |\, \{i : s_j(ghost.i) \;\wedge\; \neg(\exists k : k \in V' \wedge i \in CT_k(s_j))\} \,| \\
&\quad T(s_j) \;=\; |\, \{i : CT_i(s_j) \text{is a non-empty tree}\} \,|
\end{aligned}
$$

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 round 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, or SC 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 rounds.

Because each round takes at most $d_s + U$ amount of time, $G(s_j)$ reaches its minimum value within $(d_s + U)N$, 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 round 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 rounds, 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 round of computation until it reaches the minimum value 0, and this decreasing procedure takes no more than N rounds 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 $\mathcal{L}.0'$ within $O(N)$ amount of time.

(b) $\mathcal{L}.0'$ is closed under system execution.

In order to prove the closure of $\mathcal{L}.0'$ under system execution, we need to prove that every action at every node in the system preserves $\mathcal{L}.0'$. Without loss of generality, let us consider an arbitrary node i in the system. Since actions C_1, C_2, and SC are not enabled at node i at every state in $\mathcal{L}.0'$, they trivially preserve $\mathcal{L}.0'$. Also, because action S_1 does not modify the value of $d.i$ and $ghost.i$, it trivially preserve $\mathcal{L}.0'$ too. Action S_2 at node i preserves $\mathcal{L}.0'$ 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) $\mathcal{L}.0'$ implies $\mathcal{L}.0$.

In order to prove that $\mathcal{L}.0'$ implies $\mathcal{L}.0$, we only need to prove that $\mathcal{L}.0'$ 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 $\mathcal{L}.0'$. If $i = r$, then $SP.i = true$ at state q_0', which contradicts the fact that q_0' is a state in $\mathcal{L}.0'$; 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 $\mathcal{L}.0'$. Therefore, $\mathcal{L}.0'$ implies $\mathcal{L}.0$.

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

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

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

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

Since $\mathcal{L}.l$ is closed under system execution, and $\mathcal{L}.(l+1)$ is defined as $\mathcal{L}.l \wedge (\forall i : i \in V' \wedge dist(i,r,G) = l \Rightarrow LH.i) \wedge (\forall i : i \in V' \wedge dist(i,r,G) > l \Rightarrow d.i > l)$, we only need to prove that starting at an arbitrary state in $\mathcal{L}.l$, 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 $\mathcal{L}.l$, 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 round of system computation.

- Second, $\mathcal{L}.l \wedge (\forall i : i \in V' \wedge dist(i,r,G) = l \Rightarrow LH.i)$ 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 $\mathcal{L}.l$.

(IV) $\mathcal{L}.D \equiv \mathcal{L}$

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 $\mathcal{L}$.

□

Theorem 1 and 2 imply

Corollary 2 (Termination) *Starting at an arbitrary state, every system computation terminates at a state in $\mathcal{L}$.*

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 $\mathcal{L}$.

□

From Corollary 2, we know that LSRP guarantees the formation of the shortest path tree in a system when it starts at an arbitrary state.

Now, let us look at the property of local stabilization in a system where LSRP is used, by which the shortest path tree is guaranteed to be formed within time proportional to the perturbation size at the initial system state, and the range of contamination is proportional to the perturbation size.

When there is only one perturbed region at the initial state, we have

Lemma 1 (Local stabilization of a single perturbed region) *Starting at an arbitrary state q_0 where there is only one perturbed region, every system computation reaches a state in $\mathcal{L}$ within $O(P(q_0))$ time, and the range of contamination is $O(P(q_0))$.*

Proof: Let p be the perturbation size $P(q_0)$ for the system at state q_0.

Let R_p be the perturbed region, and p_d be the diameter of the subgraph of G on R_p.[7] The set of nodes that are not in R_p is called the *healthy region* R_h[8]. 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 protocol, the corruption of $p.i$ at node i does not contaminate 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 contaminate 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:

(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 fault propagation at state q_0; it is also possible that there exist some healthy boundary nodes that are sources of fault propagation at q_0.

(a) If no healthy boundary node is a source of fault 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 shortest path routing tree (as shown in Figure 5).

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.

[7] In shortest path routing, the number of nodes in the perturbed region is equal to $P(q_0)$.

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

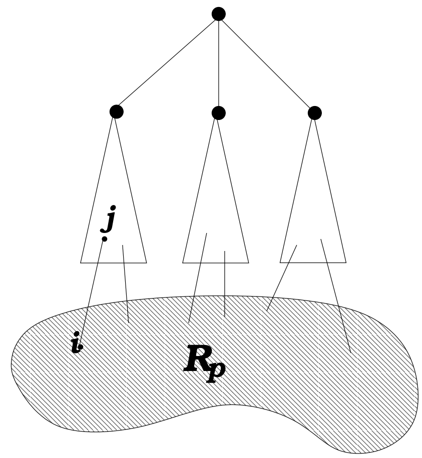


Figure 5: The "distance value" of every perturbed node is no less than that of all the healthy boundary nodes, and there is no source of fault propagation.

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 round 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 round of system computation. Therefore, starting at state q'_0, function f reaches its minimum value 0 within $O(p_d)$ rounds of system computation. When $f = 0$, there is no perturbed node any more and the system is at a state in $\mathcal{L}$. Because each round of computation takes at most time $(d_s + U)$ amount of time, the system reaches a state in $\mathcal{L}$ within $O(p_d)(d_s + U) = O(p_d)$ amount of time.

Therefore, the system reaches a state in $\mathcal{L}$ within $O(p) + O(p_d) = O(p)$ amount of time.

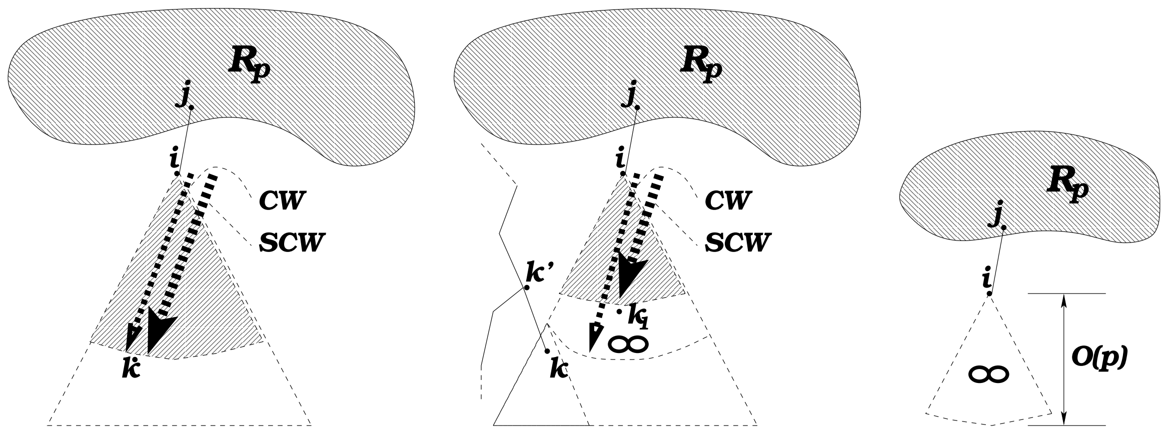
(b) If there are some non-root[9] healthy boundary nodes that are sources of fault propagation at state q_0, containment waves can be initiated at those nodes, and the perturbed region R_p can expand outward to the healthy region R_h. But this expansion is bounded.

Let us first consider the case where there is only one such non-root healthy boundary node i that is a source of fault propagation at q_0 (see Figure 6[10]). 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 fault 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

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

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

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:



(a) (b) (c)

Figure 6: The "distance value" of every perturbed node is no less than that of all the healthy boundary nodes, and there are some sources of fault propagation.

i. If action C_2 has not been executed at node i before the moment that is one round after all the nodes in S, the original set of perturbed nodes at state q_0, have converged to a legitimate state (see Figure 6(a)(b)), the farthest distance (in terms of hops) the containment wave can propagate is $O(p)$, and the system reaches a state in $\mathcal{L}$ 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 fault 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 fault 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 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 $\mathcal{L}$ 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, action S_2 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 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 6(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.

  Because the propagation speeds of different diffusing waves are controlled by introducing delays in action execution when a node schedules its enabled actions, and there is difference between clock rates at different nodes, $d_c > \alpha \cdot d_{sc}$ should hold in order for the super-containment wave to catch up with the containment wave. Under this condition, 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 $(U + d_{sc})p_{cw} + O(p) = (L + d_c)p_{cw}$ holds, which implies that $p_{cw} = \frac{O(p)}{d_c - d_{sc} + L - U} = O(p)$.[11]

  Therefore, the range of contamination and the time taken for the system to stabilize to a state in $\mathcal{L}$ 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 6(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

[11] $d_c - d_{sc}$ should be larger than $U - L$.

$O(p)$.

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 $\mathcal{L}$ within time $O(p)$ in this case.

ii. If action C_2 has been executed at node i before the moment that is one round 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 6(c)), the farthest distance the containment wave can propagate is still $O(p)$, and the system reaches a state in $\mathcal{L}$ 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 $\mathcal{L}$ within time $O(p)$, starting at state q_0'. Therefore, starting at state q_0', the system reaches a state in $\mathcal{L}$ within time $O(p)$.

If there exist multiple healthy boundary nodes that are sources of fault 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[12].

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 5 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 $\mathcal{L}$ 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 fault propagation at state q_0; it is also possible that there exist some healthy boundary nodes that are sources of fault 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 fault propagation at state q_0 (see Figure 7), 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

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

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.

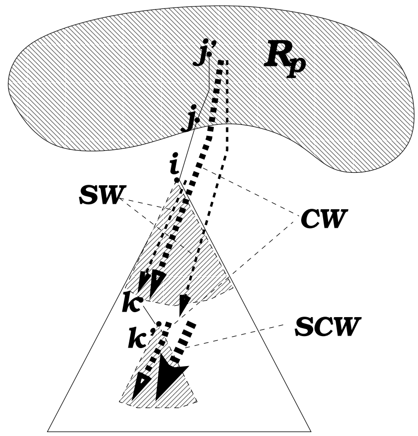


Figure 7: The "distance value" of some or all perturbed nodes is less than that of some or all healthy boundary nodes, and there is no source of fault propagation.

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. Due to the fact that, in a distributed system, the propagation speeds of different diffusing waves are controlled by introducing delays in action execution when a node schedules its enabled actions, and there is difference between clock rates at different nodes, $d_s > \alpha \cdot d_c$ should hold in order for the containment wave to catch up with the stabilization wave. Under this condition, 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 $(U + d_c)p_{sw} + O(p) = (L + d_s)p_{sw}$ holds, which implies that $p_{sw} = \frac{O(p)}{d_s - d_c + L - U} = O(p)$.[13] 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

[13] $d_s - d_c$ should be larger than $U - L$.

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 fault 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 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 $O(p)$[14].

Same as that in Ib, if there are multiple healthy boundary nodes that are not sources of fault 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 $\mathcal{L}$ are $O(p)$ in this case.

(b) If there are some non-root healthy boundary nodes that are sources of fault 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 fault propagation at state q_0, and the reasoning carries to the case where there are multiple healthy boundary nodes that are sources of fault propagation at q_0.

Since node i is a source of fault 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 8(a)), then node i will initiate a super-containment wave after all the 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 8(b)), then action S_2 will be executed at node i, and a stabilization wave be initiated at i. After action S_2 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 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.

[14] The reasoning is the same as that in Ib

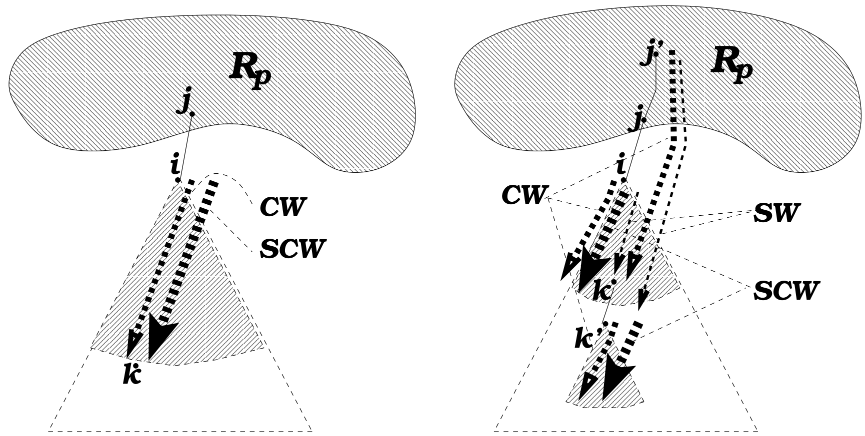


Figure 8: The "distance value" of some or all perturbed nodes is less than that of some or all healthy boundary nodes, and there are some sources of fault propagation.

In this case, there can also exist some healthy boundary nodes that are not sources of fault 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 5 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.

$\square$

When the set of perturbed nodes are not contiguous, there are multiple perturbed regions (denoted as $S_0, S_1, \ldots, S_m$, $m \geq 1$) in the system. Then, we have

Lemma 2 (Local stabilization when containment regions are disjoint) *Starting at an arbitrary state q_0 where the perturbed regions are $S_0, S_1, \ldots, S_m$ and their containment regions are disjoint, every system computation reaches a state in $\mathcal{L}$ within $O(\max_{i \in 0..m} |S_i|)$ time, and the range of contamination is $O(\max_{i \in 0..m} |S_i|)$.*

Proof: Let us consider any two perturbed regions S_i and S_j ($i \neq j$). Since their containment regions are disjoint, there exists no interference between S_i and S_j during stabilization in the sense that actions being executed at nodes in one containment region do not affect actions being executed in another containment region. In this case, the stabilization of S_i and S_j are independent of each other. Therefore, the time taken for a perturbed region S_i to stabilize as well as the diameter of its containment region is $O(|S_i|)$ according to Lemma 1.

Moreover, actions are executed in parallel at different containment regions in the system. Therefore, the time taken for the system to stabilize to a state in $\mathcal{L}$ as well as the range of contamination during stabilization only depends on that of the perturbed region the diameter of whose containment region is the largest. That is, the time taken for the system to stabilize to a state in $\mathcal{L}$ as well as the range of contamination during stabilization is $\max_{i\in 0..m} O(|S_i|)$, which equals to $O(\max_{i\in 0..m} |S_i|)$.

□

Given any two perturbed regions S_i and S_j $(i \neq j)$ at a state q, the half-distance between S_i and S_j is half of the minimum distance from a node in S_i to another node in S_j, that is, $\min_{k\in S_i, k'\in S_j} \lfloor \frac{dist(k,k',G.q)}{2} \rfloor$, where $dist(k, k', G.q)$ denotes the minimum distance between node k and k' in graph $G.q$. Then, Lemma 2 implies

Corollary 3 *Starting at an arbitrary state q_0 where the perturbed regions are $S_0, S_1, \ldots, S_m$ and the half-distance between any two of them is $\Omega(\max_{i\in 0..m} |S_i|)$, every system computation reaches a state in $\mathcal{L}$ within $O(\max_{i\in 0..m} |S_i|)$ time, and the range of contamination is $O(\max_{i\in 0..m} |S_i|)$.*

Lemma 3 (Local stabilization when containment regions are adjoining) *Starting at an arbitrary state q_0 where the perturbed regions are $S_0, S_1, \ldots, S_m$ and their containment regions are adjoining, every system computation reaches a state in $\mathcal{L}$ within $O(\sum_{i=0}^{m} |S_i|)$ time, but the range of contamination is still $O(\max_{i\in 0..m} |S_i|)$.*

Proof: Let us consider any two perturbed regions S_i and S_j $(i \neq j)$ whose containment regions are adjoining. 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[15], there is still no interference between S_i and S_j in the sense that the size of the containment region of one perturbed region in the presence of the other adjourning containment region can be no larger than that in the absence of the adjourning containment region. Therefore, the range of contamination during stabilization is still the diameter of the largest containment region in the system, i.e., $O(\max_{i\in 0..m} |S_i|)$. And this is the same as the case where the containment regions of all the perturbed regions are disjoint.

However, given any two perturbed regions S_i and S_j $(i \neq j)$ whose containment regions are adjoining, the stabilization of one perturbed region (e.g. S_i) may depend on the stabilization of the other perturbed region (e.g. S_j), due to the fact that the stabilization in shortest path routing is essentially a diffusing computation from the destination node. That is, one perturbed region can stabilize only after the other perturbed region has stabilized. Therefore, in the worst case where the set of perturbed regions in the system stabilize sequentially, the time taken for the system to stabilize is the summation of the stabilization time for each perturbed regions, i.e., $\sum_{i=0}^{m} O(|S_i|)$ which equals to $O(\sum_{i=0}^{m} |S_i|)$.

□

Lemma 2 and 3 imply

Corollary 4 (Local stabilization of multiple perturbed regions) *Starting at an arbitrary state q_0 where the perturbed regions are $S_0, S_1, \ldots, S_m$, every system computation reaches a state in $\mathcal{L}$ within $O(P(q_0))$ time, and the range of contamination is $O(\max_{i\in 0..m} |S_i|)$ (which is $o(P(q_0))$).*

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

Lemma 1 and Corollary 4 imply

Theorem 3 (Local stabilization) *Starting at an arbitrary state q_0, every system computation reaches a state in $\mathcal{L}$ within $O(P(q_0))$ time, and the range of contamination is $O(\mathrm{MAXP})$, where MAXP denotes the number of nodes in the largest perturbed region at q_0 and is $o(P(q_0))$. That is, the system is $\mathcal{F}$-local stabilizing, where $\mathcal{F}$ is a linear function.*

By Theorem 3, we see that LSRP solves the shortest path routing problem in a linear-local stabilizing manner.

In a distributed system, faults may keep occurring in certain regions of the system with some frequencies or for certain periods of time. For example, congestion-induced route flapping may keep occurring at network edge routers for a certain period of time in Internet inter-domain routing [18]. Due to its property of local stabilization, LSRP tolerates these kinds of faults while still guaranteeing local fault containment and local stabilization, as long as the frequency of repeated faults is not too high or the period of continuous faults is not too long. Theorem 3 implies

Corollary 5 (Local fault containment in spite of repeated faults) *If some transient fault that drives a system G to state q_0 keeps occurring, but the interval between two consecutive occurrences of the fault is $\Omega(P(q_0))$, then the range of contamination is $O(\mathrm{MAXP})$, where MAXP denotes the number of nodes in the largest perturbed region at q_0 and is $o(P(q_0))$.*

Also, we have

Theorem 4 (Local stabilization in spite of faults during stabilization) *Starting at an arbitrary state q_0 at time t_0, if only transient faults occur, they occur only inside containment regions, and no fault occurs after time $t_0 + T$, then every system computation reaches a state in $\mathcal{L}$ within $O(P(q_0) + T)$ time, and the range of contamination is $O(P(q_0) + T)$.*

Proof: This claim follows from the proofs for Lemma 1 and Corollary 4, and the fact that if, during stabilization, no fault occurs outside containment regions in the system and no fault occurs after T amount of time since the system computation starts, the maximum distance (in terms of the number of hops) that a "mistakenly" initiated stabilization wave or containment wave from a healthy boundary node at q_0 can propagate is at most $O(T)$ more than that for the case where no faults occur in the system since the system computation starts.

□

From Theorem 4, we see that, starting at q_0, the time taken for a system to stabilize and the range of contamination are still $O(P(q_0))$, if faults stop occurring in the containment regions $O(P(q_0))$ time after the system computation starts and no faults occur to nodes outside the containment regions.

5.2 Properties of loop freedom and quick loop removal

Theorem 5 (Loop freedom) *Starting at an arbitrary state where there is no loop, every system computation reaches a state in $\mathcal{L}$, and there is no loop at any state along the computation.*

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[16] 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

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

as $v_0, v_1, \ldots, v_{l-1}$ with $p.v_i = v_{i-1}$ $(i = 0, 1, \ldots, l-1)$[17] at state q_1.

Then the loop L must have been in the system at the state q'_1 that is 1-round prior to state q_1 (i.e., q_1 is the system state after a round of system computation that starts at state q'_1). 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 action S_2 or C_2 must be executed at node i in the round of computation between state q'_1 and q_1, which implies that $d.v_{i-1} < d.v_i$ at state q'_1.

- Let $S = \{k : k \in L \wedge$ action S_2 is executed at node k in the round 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 fault 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 round 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 fault propagation at q'_1 (i.e., $MP.v_j \vee SP.v_j = false$ at q'_1), then action S_2 must be executed at node v_j in the round 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.

□

From Theorem 5, we see that there is no loop in the system during stabilization if the only possible fault in a system is node fail-stop, because no loop can be formed just by node fail-stop, and there is no loop at any initial state of a system computation if the only fault is node fail-stop.

Theorem 6 (1-round loop breakage) *Starting at an arbitrary state where there exists at least one loop, every system computation reaches a state where there is no loop after at most one round of computation.*

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 round 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 $p.v_i = v_{i-1}$ $(i = 0, 1, \ldots, l-1)$[17].

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 fault propagation at q_0 (i.e., $MP.v_i \vee SP.v_i = true$ at q_0), then either

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

action S_1 or action C_1 will be executed within one round 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 fault propagation at q_0 (i.e., $MP.v_i \vee SP.v_i = false$ at q_0), then action S_2 must be executed at node v_i within one round 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 5.

(II) Second, there is no new loop formed in any round of system computation. This trivially follows from the proof for Theorem 5.

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

□

Theorem 5 and 6 imply

Corollary 6 (Quick loop removal) *Starting at an arbitrary state, there will be no loop in the system after at most one round of computation.*

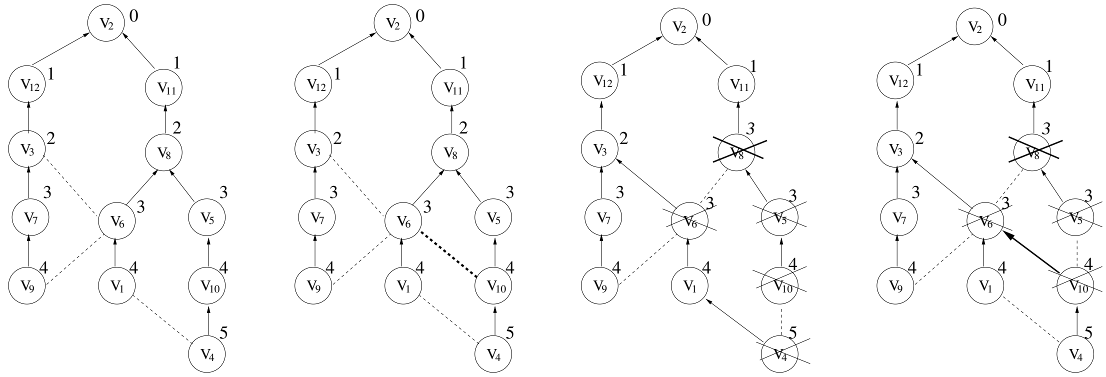
6 Impact of network topology on local stabilization

For the problem of shortest path routing, the network topology of a system can affect the perturbation size, the range of contamination, and the self-stabilization time.

Given a system topology $G.q_0(V.q_0, E.q_0)$ at state q_0, if we add some edges to $G.q_0$ and obtain another system topology $G.q'_0(V.q'_0, E.q'_0)$ at state q'_0 with denser edges (i.e. $E.q_0 \subset E.q'_0$), then for every node that is both in $G.q_0$ and in $G.q'_0$, the number of different shortest paths to the destination node in $G.q_0$ will be no more than that in $G.q'_0$. Then, if the same node i fail-stops in both $G.q_0$ and $G.q'_0$, and $G.q_0$, $G.q'_0$ transit to $G.q$, $G.q'$ respectively, the number of nodes that are perturbed due to the fail-stop of i in $G.q$ will be no less than that in $G.q'$. More generally, if the same faults occur when the system is at q_0 and at q'_0, and the system reaches q and q' respectively after the faults, the perturbation size at state q will be no less than that at q'. Moreover, even if the perturbation sizes at q and q' are the same, the potential mistakenly initiated containment wave will propagate no farther in $G.q'$ than in $G.q$ as shown in Figure 6(b). Therefore, the time taken for the system to stabilize from q and the range of contamination during stabilization are no less than that with respect to q'. Formally, this fact is presented in Proposition 5.

Proposition 5 (Higher edge density is conducive to local stabilization) *Given a system G and two system states q and q' such that $V.q = V.q'$, $E.q \subseteq E.q'$, and $(\forall i : i \in V.q \Rightarrow q(i) = q'(i))$, then $P(q) \geq P(q')$, $R_c(q) \geq R_c(q')$, and the time taken for G to stabilize from q is no less than that with respect to q'.*

One example is shown in Figure 9. Figure 9(a) and (b) represent system topology $G.q_0$ and $G.q'_0$ at legitimate states q_0 and q'_0 respectively. The only difference between $G.q_0$ and $G.q'_0$ is that edge $(v_6,\ v_{10})$ is not in $G.q_0$ but is in $G.q'_0$. Then,



(a) System topology $G.q_0$

(b) System topology $G.q'_0$

(c) Perturbed state in $G.q_0$

(d) Perturbed state in $G.q'_0$

Figure 9: An example where denser edges reduce the perturbation size and range of contamination upon perturbations in the system

- If node v_8 fail-stops at both state q_0 and q'_0, and the system reaches state q and q respectively, then the perturbation size at state q is 4 (i.e., nodes v_6, v_5, v_4, and v_{10} have to change their next-hops), whereas the perturbation size at q' is 3 (i.e., nodes v_6, v_5, and v_{10} have to change their next-hops, but v_4 does not);
- If the state of v_8 is corrupted in both $G.q_0$ and $G.q'_0$ such that $d.v_8 = 3$, then the perturbed states in $G.q_0$ and $G.q'_0$ during stabilization can be those as shown in Figure 9(c) and (d) respectively, by which we can see that the range of contamination in $G.q_0$ can be 3, whereas the range of contamination in $G.q'_0$ can only be 2 at most. The reason for this is that action S_2 is enabled at v_{10} in $G.q'_0$ and the mistakenly initiated containment wave from v_5 will not propagate beyond v_{10}. Similarly, the time taken to stabilize in $G.q'_0$ is no more than that in $G.q_0$.

In wireless networks, especially in wireless sensor networks [19], 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). Our conclusion, therefore, is that wireless (sensor) networks with higher edge density are likely to contain faults more tightly and to stabilize faster.

7 Related work

In [2], a self-stabilizing algorithm was proposed for maintaining a spanning tree in a network. Time-optimal versions were proposed later [1, 3]. None of these contain faults locally or achieve local stabilization. In [19], we proposed a local stabilizing protocol GS^3 for clustering as well as shortest path routing in wireless networks whose nodes lie in a planar and whose density is high. The differences in model assumptions make LSRP and GS^3 incomparable.

A loop-free distance-vector based protocol DUAL was proposed in [4] and incorporated into EIGRP [8]. DUAL does not achieve local stabilization: unnecessary global diffusing computations can be introduced in DUAL when local transient perturbations, such as congestion, occur. This phenomenon becomes worse when networks are under stress, wherein transient faults happen more frequently for some period of time [18]. Furthermore, in DUAL, if a loop is already formed (e.g., due to state corruption), the time taken to break the loop is $\theta(L)$, where L is the length of the loop; in contrast, a loop can be broken within a small constant amount of time in LSRP, irrespective of the loop length.

In [5] and [7], algorithms were proposed to contain a single state-corruption during stabilization of a spanning tree; these algorithms do not deal with multiple faults or node fail-stops. In [6], a fault-containing self-stabilizing algorithm was proposed for a consensus problem, but it considers only a linear topology and its range of contamination can be exponential in the perturbation size; also, the algorithm does not apply to the problem of shortest path routing. In [10], an algorithm was proposed for mending faults locally. But the representation of a solution is replicated across the whole system, therefore it is not local in space, and furthermore local stabilization is not guaranteed.

In [5] and [14], the concept of fault containment was proposed. However, in [5], 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. In [14], it requires the range of contamination to be constant , which is too strict to be applied to such problems as routing, where the locality of problems[18] is not constant. Our concept of $\mathcal{F}$-local stabilization focuses on the amount of work needed during stabilization, and it is general enough to be applied to such problems as routing.

8 Concluding remarks

In this paper, we have formulated the concepts of perturbation size, $\mathcal{F}$-local stabilization, and range of contamination, in order to formally characterize local stabilization properties of networked and distributed systems. These concepts are generically applicable to networked and distributed systems, and are thus interesting in their own right.

For the problem of local stabilization in shortest path routing, we designed LSRP. One technique used in LSRP to achieve local stabilization is to layer the diffusing computation in shortest path routing into multiple diffusing waves; in particular, the stabilization wave, the containment wave, and the super-containment wave, all of which are locally confined around the perturbed regions. The stabilization wave guarantees the eventual convergence of a system to a legitimate state, the containment wave prevents a mistakenly initiated stabilization wave from propagating far away from where faults have occurred, and the super-containment wave prevents a mistakenly initiated containment wave from propagating far away. An interesting point in the wave-layering is that a node which has mistakenly initiated a diffusing wave w_1 (e.g., a stabilization wave or a containment wave) will detect this fact within time proportional to the perturbation size, and will initiate another diffusing wave w_2 (e.g., a containment wave or a super-containment wave) that will catch up with and stop w_1 before it propagates far away. Moreover, there is no potential fault introduced by the super-containment wave, therefore we do not need another diffusing wave to contain potential faults in super-containment wave, which enables the concept of layering. Another technique adopted to enable

[18]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.

local stabilization is to avoid forming loops during stabilization, which would delay the convergence, and to remove transient loops quickly if there are any in the initial state.

In LSRP, we have introduced delays in action execution to control the propagation speeds of diffusing waves. This does not 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, which is especially desirable in large-scale systems where faults generally occur only at a small part of the system. Moreover, the method of introducing delays in action execution is also commonly used in Internet routing in order to reduce control overhead and routing flaps. For example, in BGP the timer MinRouteAdvertisementInterval is used to control the frequency of route exchange between BGP peers. The timer is similar to the delay introduced for the stabilization wave in LSRP. Therefore, in implementing LSRP, we only need to introduce smaller timers for the containment wave and super-containment wave.

For the problem of shortest path routing, we have observed that higher edge density in a system can reduce the perturbation size, the range of contamination, and the self-stabilization time. This leads to an interesting question for network design as well as for self-configuration in (wireless) networks, namely, how to design or self-configure a network such that the perturbation size, the range of contamination, and the self-stabilization time are minimized.

In the literature of network routing protocol design [8], formation of routing loops is regarded as problematic, and a variety of schemes have been proposed to avoid forming loops. For example, in intra-domain routing, the distance-vector protocol DUAL [4] introduced the concept of "feasibility conditions for loop freedom"; yet, the presently used protocol OSPF abandons the distance-vector paradigm to avoid loop formation, even though its approach does not scale well[19]. And in inter-domain routing, BGP [16] uses the concept of AS_PATH (i.e., the sequence of ASs in the path from one AS to another AS) to avoid forming loops. However, fault propagation and routing instability remain as problems in OSPF and BGP. The root cause appears to be that these protocols are not designed to be tolerate such faults as misconfiguration and persistent congestions, which are special kinds of state corruption. In our protocol LSRP, state corruption is dealt with by way of local stabilization. As a result, looping is implicitly avoided by taking loop-formation as a kind of state corruption, without introducing special mechanisms to deal with potential loops. By local stabilization, LSRP prevents faults from propagating far away and increases the stability as well as availability of a system. Therefore, the question of whether we should take various kinds of faults as state corruption and deal with them by way of (local) stabilization deserves further exploration.

In this paper, we have assumed that the weight of each edge is 1 for simplicity of presentation, but our protocol can be applied with minor modification to scenarios where the weights are arbitrary non-negative values. This will be incorporated into our future work where we study the application of our protocol to existing distance-vector (and path-vector) based routing protocols, such as DSDV, AODV, RIP, and BGP. The more detailed study of the impact of network topology on local stabilization in shortest path routing is a subject for our future work. We will also study how to achieve local stabilization without using security mechanisms (such as authentication) in the presence of byzantine faults.

[19]In OSPF, the maximum recommended number of routers per OSPF area is 200 [8].

Acknowledgment

We thank Mohamed Gouda, Ted Herman, and Shay Kutten for their helpful comments.

References

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

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

[3] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese. Time optimal self-stabilizing synchronization. In *ACM STOC 1993*.

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

[5] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *ACM PODC 1996*.

[6] S. Ghosh and X. He. Scalable self-stabilization. In *IEEE ICDCS'WSS 1999*.

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

[8] C. Huitema. *Routing in the Internet*. Prentice-Hall, Inc., 1999.

[9] M. Jayaram and G. Varghese. Crash failures can drive protocols to arbitrary states. In *ACM PODC 1996*.

[10] S. Kutten and D. Peleg. Fault-local distributed mending. In *ACM PODC 1995*.

[11] C. Labovitz, A. Ahuja, R. Wattenhofer, and S. Venkatachary. The impact of internet policy and topology on delayed routing convergence. In *IEEE INFOCOM 2001*.

[12] C. Labovitz, G. R. Malan, and F. Jahanian. Origins of internet routing instability. In *IEEE INFOCOM 1999*.

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

[14] M. Nesterenko and A. Arora. Local tolerance to unbounded byzantine faults. In *IEEE SRDS 2002*.

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

[16] Y. Rekhter and T. Li. A border gateway protocol 4 (bgp-4). *RFC 1771, IETF*, 1995.

[17] A. Shaikh, L. Kalampoukas, R. Dube, and A. Varma. Routing stability in congested networks: Experimentation and analysis. In *ACM SIGCOMM 2000*.

[18] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. Observation and analysis of bgp behavior under stress. In *ACM SIGCOMM Internet Measurement Workshop 2002*.

[19] H. Zhang and A. Arora. GS^3: Scalable self-configuration and self-healing in wireless networks. In *ACM PODC 2002*.