LSRP: Local Stabilization in Shortest Path Routing

Anish Arora, *Member, IEEE*, and Hongwei Zhang, *Student Member, IEEE*

***Abstract*— 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 each having a different propagation speed, 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 the mistakenly initiated stabilization wave, the super-containment wave contains the mistakenly initiated containment wave, and the super-containment wave self-stabilizes itself locally. Second, LSRP avoids forming loops during stabilization, and it removes all transient loops within small constant time. To the best of our knowledge, LSRP is the first protocol that achieves local stabilization in shortest path routing.**

***Index Terms*— Shortest path routing, local stabilization, perturbation size, range of contamination, containment region**

I. 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 [1], [2].

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 [3], [2], [4], it is desirable that faults be contained locally around the regions where they have occurred, and that the time taken for a system to stabilize is 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 protocols.

Distance-vector (and its variant, path-vector) protocols for the Internet, such as Routing Information Protocol (RIP) and BGP, have long been studied [5]. 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 [6]. 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 [2], and result in severe instability [7], [2]. 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 [1], [8], [2]. 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.

Related work. The concept of fault containment is proposed in [10], [11], and [3]. Nevertheless, [10] and [11] only consider 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 protocol actions executed) needed for a system to stabilize is a function of the perturbation size; [3] only considers the case where the impact of every fault is within constant distance from where it occurs, which is too strict to be applied to problems such as routing, where the locality of the problem[1] is not constant.

A locally stabilizing protocol GS^3 is proposed in [4] for clustering as well as shortest path routing in wireless sensor networks where nodes are densely distributed. To achieve local stabilization, GS^3 takes advantage of the high node distribution density and the geographic information on node distribution; the sensor network model assumed by GS^3 makes

An extended abstract containing some preliminary results of this paper appeared in 2003 IEEE-IFIP International Conference on Dependable Systems and Networks (DSN-2003). 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.

The authors are with the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210, USA (e-mail: {anish, zhangho}@cis.ohio-state.edu).

[1] We regard the locality of a problem as the maximum minimum distance between any two nodes that have to be involved in the definition of the problem.

it inapplicable to other general networks such as the Internet. In [10], algorithms are proposed to contain a single state-corruption during stabilization of a spanning tree, but these algorithms do not deal with multiple faults and the fail-stop of nodes.

In [11], a broadcast protocol is proposed to contain externally observable variables in the presence of state corruptions, but the protocol allows for global propagation of internal variables. In [12], a fault-containing self-stabilizing algorithm is proposed for a consensus problem, but it only considers linear network topologies and the distance faults propagate can be exponential in the perturbation size; and the algorithm does not apply to the problem of shortest path routing.

Loop-free distance-vector protocols DUAL [13] and LPA [14] are proposed for Internet routing. Nonetheless, neither DUAL nor LPA guarantees local stabilization: faults can propagate globally in DUAL as well as in LPA when local transient perturbations, such as congestion and state corruption, occur. This phenomenon becomes worse when networks are under stress, with transient faults happening more frequently for some period of time [2]. Furthermore, the time taken to break a loop which has already formed (e.g., due to state corruption) is not constant in DUAL and LPA; instead it is proportional to the length of the loop.

Contributions of the paper. To characterize the properties of locally stabilizing systems, we formulate the concepts of perturbation size, $\mathcal{F}$-local stabilization, and range of contamination. These concepts take into account the minimum amount of work required for systems to stabilize and are generically applicable to networking as well as distributed computing problems.

We also design LSRP (for Locally 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 not only guarantees loop-freedom during stabilization, it also removes any existing loop (which is created by a fault) within constant time irrespective of the loop length.

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 II, we present the system, fault, and computation model. In Section III, we define local stabilization, and analyze the properties of locally stabilizing systems. We present our LSRP protocol that solves the problem of local stabilization in shortest path routing in Section IV, and analyze its properties in Section V. In Section VI, we discuss the impact of network topology on local stabilization. We also discuss issues related to the application of LSRP. We make concluding remarks in Section VII.

II. 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, W), where V and E are the set of nodes and the set of edges in the system respectively, and W is a positive function that defines the weight of each edge in E. (W is also called the *weight function* hereafter.) 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 each edge $(i, j) \in E$, its weight is denoted by $w.i.j$.

There is a clock at each node. The ratio of clock speeds between any two neighboring nodes in the system is bounded from above by α, but no extra constraint on the absolute values of clocks is enforced.

Message transmission between nodes is reliable, and message passing delay along an edge is bounded from above and from below by U and L respectively.

Protocol notation. We write protocols using a variant of the Abstract Protocol notation [15]. At each node, the protocol consists of a finite set of variables and actions. Each action consists of three parts: guard, guard hold-time, and statement. For convenience, we associate a unique name with each action. Thus, an action has the following form:

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

The guard is either a boolean expression over the protocol variables of the node or a message reception operation, d is the guard hold-time ($d \geq 0$), and the statement updates zero or more protocol variables of the node and/or sends out some message(s). If $d = 0$, we write the action in the following form:

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

For an action whose guard is a message reception operation, its guard hold-time must be 0.

For an action named a, its guard hold-time is denoted by $d.a$. An action a is enabled at time t if the guard of a evaluates to true at t. An action a is executed at time t only if a is continuously enabled from time $(t - d.a)$ to t. 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, the state of a node, i.e., the values of all the variables of the node, can be corrupted, and the weight function can change.

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 reflect changes in system topology as well as weight function, we regard the state of G as the union of the current system topology, the current weight function, the state of all the up nodes, and the message(s) in the up edges (i.e., the messages that are sent but not yet received). At a system state q, the system topology, the weight function, and the state of an up node i are denoted as $G.q(V.q, E.q)$, $W.q$, 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 (i) for every $k \geq 1$, $t_k \leq t_{k+1}$, and each state transition $q_{k-1}, (a_k, t_k), q_k$ means that the execution of action a_k at time t_k changes the system state from q_{k-1} to q_k; and (ii) 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 actions of 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 is a minimal computation segment γ that starts at a state q_k $(k \geq 0)$ and, in γ, (i) every up node that has an action a continuously enabled from some time t' to $(t' + d.a)$, where $t' \leq t_k$ and $(t' + d.a) \geq t_k$, executes at least one action, and (ii) if a message is sent to a node i, the action that receives the message must be executed at i. (We assume t_0 is the time when β starts.)

III. Local stabilization: concepts and properties

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

A. Concepts related to local stabilization

In a distributed system, the variables that each node needs to maintain depend both on the problem and on 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. In the problem of shortest path routing, for instance, every node has to maintain the distance and the next-hop on its chosen shortest path to each destination: maintaining the distance is necessary for a node to coordinate with others to find its shortest path to the destination, and maintaining the next-hop is necessary for a node to forward packets to the destination. Therefore, the variables used to record the distance and the next-hop are inherent in the problem of shortest path routing. We call variables that are inherent in the problem *problem-specific variables*. At a system state q, the value of the set of problem-specific variables at a node i is denoted as $q(i.p)$.

Dependency among nodes & edges. Given a problem, a node may depend on another node or edge in a distributed system, because, when faults occur to the latter, the former may have to change the values of its problem-specific variables in order for the system to converge to a legitimate state (i.e., to stabilize), 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 depends 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.

In general, if some up nodes in a system have to adapt the values of their problem-specific variables in order for the system to stabilize (irrespective of the state to which the system stabilizes) after a set of nodes and edges fail-stop while the system is at a legitimate state, we regard these up nodes as dependent upon the fail-stopped nodes and edges. Similarly, if some existing nodes in a system have to adapt the values of their problem-specific variables after a set of nodes and edges newly join the system while it is at a legitimate state, we regard these existing nodes as dependent upon the newly-joining nodes and edges; for convenience, we also regard the newly-joining nodes as dependent on themselves, since they need to adapt the values of their problem-specific variables too.

Formally, given a set of nodes V', a set of edges E', and a legitimate state q, we define the *dependent set of* V' *and* E' *at* q, denoted by $D_q(V', E')$, as:

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

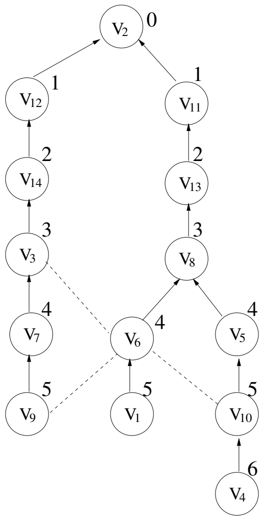
where G_- is the system topology after V' and E' have fail-stopped, and G_+ is the system topology after V' and E' newly join the system, i.e., $G_- = (V.q \setminus V', E.q \setminus E')$, $G_+ = (V.q \cup V', E.q \cup E')$.[2] By definition, the dependent set $D_q(V', E')$ denotes the minimum set of nodes that are affected when the set of nodes V' and the set of edges E' fail-stop or newly join the system while it is at state q. (Note that the definition also applies to changes in link weight, since the weight change at a link can be regarded as the fail-stop of the link with the old weight followed by the join of the link with the new weight.)

Considering the problem of shortest path routing, for example, Figure 1 represents a legitimate state q. If node v_{11} and edge (v_2, v_{12}) fail-stop at q, all the other nodes except for v_2 in the system need to invalidate their distance values as well as their next-hops on their paths to v_2, since there exists no route from any node to v_2 any more. Thus, $D_q(\{v_{11}\}, \{(v_{12}, v_2)\}) = \{v_1, v_3, \ldots, v_{10}, v_{13}, v_{14}\}$. Similarly, if the edge (v_2, v_8) joins at q, node v_9 and the nodes in the subtree rooted at v_8 need to change their distance values.[3] Thus, $D_q(\emptyset, \{(v_2, v_8)\}) = \{v_8, v_6, v_5, v_9, v_1, v_{10}, v_4\}$.

Perturbation size. Therefore, a node j can be affected by a fault in two ways irrespective of the protocols used: j is directly affected by a state corruption which occurs to j itself, and j is indirectly affected by a non-state-corruption fault

[2] The node set V' and edge set E' should be such that G_- and G_+ are valid graphs.

[3] Note that nodes v_8 and v_9 also need to change their next-hops.



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. For clarity of presentation, we assume that the weight of each edge is 1.

Fig. 1. A legitimate system state

(such as fail-stop) which occurs to a node or an edge that j depends on. Then, corresponding to each set of faults that leads a system to an illegitimate state q, there is a set of nodes in $V.q$ that are affected either directly or indirectly by the faults, and the number of affected nodes denotes the degree of perturbation by the faults. Given an illegitimate state q, it could have been reached in different ways (i.e., from different legitimate states by different sets of faults), thus the number of affected nodes at q depends on how the system reaches q by certain faults. To characterize the minimum amount of work required to recover from the perturbation at q, we define the *perturbation size at* q as the minimum number of affected nodes at q considering all the possible ways q could have been reached. Formally,

Definition 1 (Perturbation size): The perturbation size at a system state q, denoted as $P(q)$, is $\min_{q' \in Q_l} |A_{q'} \cup B_{q'}|$ where

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

If q is reached from a legitimate state q' by some faults, $A_{q'}$ in the above definition denotes the set of nodes where state corruption occurred, and $B_{q'}$ denotes the set of nodes that depend on some other nodes where certain non-state-corruption faults occurred. Intuitively, the perturbation size at q equals to the minimum number of nodes in $V.q$ 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. Thus, it also reflects the minimum amount of work needed to correct a perturbation.

Given an illegitimate state q, there may exist two legitimate states q_1 and q_2 such that $|A_{q_1} \cup B_{q_1}| = |A_{q_2} \cup B_{q_2}| = P(q)$. Consider a consensus problem where all the nodes in a system need to take the same value, for instance, at a state q where one half of the nodes in the system take 3 and the other half take 4, there exist two legitimate states q_1 and q_2 such that every node takes 3 at q_1, every node takes 4 at q_2, and $|A_{q_1} \cup B_{q_1}| = |A_{q_2} \cup B_{q_2}| = P(q) = \frac{k}{2}$, where k is the number of nodes in the system. To reflect the above situation, we define the *set of potentially perturbed sets of nodes at state* q, denoted by $PP(q)$, as $\{A_{q'} \cup B_{q'}: q' \in Q_l \wedge |A_{q'} \cup B_{q'}| = P(q)\}$.

To illustrate the concept of perturbation size for the problem of shortest path routing, let us consider scenarios when different faults occur while a system is at a legitimate state q' as shown in Figure 1:

- If a state corruption occurs to node v_8, then the perturbation size at the state after the corruption is 1 and the set of potentially perturbed set of node is $\{\{v_8\}\}$, since only v_8 needs to change its state in order for the system to stabilize to the legitimate state, and at least one node in the system needs to change its state in order for the system to stabilize.
- If node v_8 fail-stops, then the perturbation size is 3 and the set of potentially perturbed set of 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.

Local stabilization. Based on the protocol-independent concept of perturbation size which reflects the minimum amount of work required for a system to stabilize from a state, we define the concept of $\mathcal{F}$-local stabilization which reflects the properties of protocols in the presence of faults.

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 and $\mathcal{F}$ is a linear function, we say that the system is locally stabilizing (for simplicity).

Given an $\mathcal{F}$-local stabilizing system and a system computation β that starts at a state q and reaches a legitimate state q', the *perturbed set of nodes at* q, denoted as $PN(q)$, is defined as the maximal set of nodes that are in the same potentially perturbed set of nodes at q and that change state from q to q'. Formally, $PN(q) = \{i : i \in V.q \wedge q(i) \neq q'(i)\} \cap$ S', where S' $\in PP(q)$ and $|\text{S'} \cap \{i : i \in V.q \wedge q(i) \neq q'(i)\}| = max_{\text{S} \in PP(q)}|\text{S} \cap \{i : i \in V.q \wedge q(i) \neq q'(i)\}|$.

A node i is *perturbed* at q if $i \in PN(q)$, otherwise, it is *healthy* at q. A node is *contaminated* if it is healthy at q and if the node executes at least one protocol action during stabilization. Then, the *range of contamination*, denoted by $R_c(q)$, is defined as the the maximum hop-distance from the set of contaminated nodes to the perturbed set of nodes $PN(q)$. That is,

$$R_c(q) \;=\; \max_{i \in S_c} hops(i, PN(q))$$

where

$$
\begin{aligned}
S_c = \{i : i \in V.q \wedge i \text{ is healthy at } q \wedge \text{ some protocol} \\
\text{action is executed at } i \text{ during stabilization}\}, \\
hops(i, PN(q)) = \min_{j \in PN(q)} hops(i, j, G.q), \\
hops(i, j, G.q) = \text{the number of hops in the shortest} \\
\text{path between } i \text{ and } j \text{ in } G.q.
\end{aligned}
$$

By definition, the range of contamination $R_c(q)$ denotes the distance to which the perturbation at q propagates during stabilization, thus $R_c(q)$ should be 0 ideally or be a function of the perturbation size at q in practice.

B. 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 claim comes from the observation that the time taken for a distributed algorithm to stabilize is at least proportional to the distance information propagates in the algorithm.)
- 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.
- The availability of an $\mathcal{F}$-local stabilizing system is high in the sense that it stabilizes quickly after perturbations and the impact of perturbations is contained locally around where they occur.

IV. 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 routing protocols, present the protocol concepts underlying LSRP, and finally present the design of LSRP.

A. Problem statement

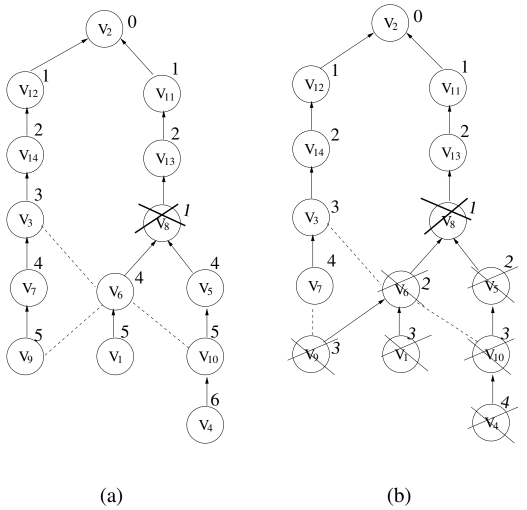
The problem is to design a protocol that, given a system $G(V, E, W)$ and a destination node $r \in V$, constructs and maintains a spanning tree T_G (called *shortest path tree*) of G that meets the following requirements:

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

B. Fault propagation in existing distance-vector protocols

Existing distance-vector routing protocols are based on the distributed Bellman-Ford algorithm [5], [16]. In these protocols, 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 $d.j + w.i.j$. 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 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 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.



(a) (b)

Fig. 2. Example of fault propagation in existing distance-vector routing protocols

C. 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 at any legitimate state;
- Second, before 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 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 v_6 and v_5 happens to the neighboring nodes of v_6 and v_5, and so on.
- Third, after detecting that its state has been corrupted, 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 is a diffusing computation by which nodes in the system learn their routes to a destination gradually. To achieve local stabilization, we design our protocol LSRP by layering the diffusing computation into three diffusing waves: the *stabilization wave*, the *containment wave*, and the *super-containment wave* (see Figure 3). Faults in a stabilization wave are contained by a containment wave, faults in a containment wave are contained by a super-containment wave, and each super-containment wave self-contains. To enable the above fault containment, containment waves propagate faster than stabilization waves, and super-containment waves propagate faster than containment waves.

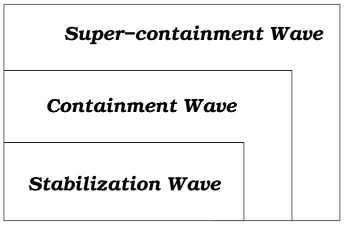


Fig. 3. Layering of diffusing waves in shortest path routing

The stabilization wave is a diffusing computation that implements the basic distributed Bellman-Ford algorithm with some changes to cooperate with the containment wave. A stabilization wave can propagate the "correction" action that enables a system to converge to a legitimate state, but a mistakenly initiated stabilization wave can propagate faults far away from where they initially occurred, as shown in Figure 2. To prevent a mistakenly initiated stabilization wave from propagating faults unboundedly, the containment wave is introduced.

Containing the stabilization wave. To enable fault containment and local stabilization, the source of fault propagation should detect the existence of the propagation and initiate a containment wave to stop it. In shortest path routing, each node chooses as its next-hop the neighbor that offers the least distance to the destination. Therefore, only "small distance value" propagates in distance-vector routing: if the distance value of a node is corrupted to be less than than it could have been (via the distance value offered by the neighbors of the node), it is highly likely that a neighboring node will propagate the corrupted value by choosing the corrupted node as its next-hop; in contrast, if the distance value of a node is corrupted to be greater than it could have been, it is less likely that a neighboring node will propagate the corrupted value.

Therefore, we regard a node as a "potential source of fault propagation" (simply called source of fault propagation hereafter) if its distance value is less than what its neighboring nodes could have provided. Formally, a node i is a source of fault propagation if, for every neighboring node j of i that is not involved in any containment wave, $d.j + w.i.j > d.i$ holds, and either i is not the destination node, or $d.i$ is not equal to 0. For example, node v_8 in Figure 2(a) is a source of fault propagation.

Whenever a node detects itself being a source of fault propagation, the node initiates a containment wave to stop the stabilization wave, if any, which has propagated the corrupted state of the node. The containment wave propagates along the same path as that by the stabilization wave. Since the containment wave propagates faster than the stabilization wave, it is able to catch up with and stop the stabilization wave.

Nevertheless, a containment wave can be mistakenly initiated due to state corruption. For example, in Figure 1, if the state of v_{11} is corrupted such that $d.v_{11} = 2$, node v_{13} will become a source of fault propagation and a containment wave will be initiated by v_{13}. To prevent a mistakenly initiated containment wave from propagating unboundedly, 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 the perturbed regions have stabilized. 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.

For the above wave-layering approach to work, the super-containment wave must self-stabilize itself locally upon perturbations; otherwise, there would be 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 [5] 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 that can select one of its descendants 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.

D. The design of LSRP

The protocol LSRP (Locally 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.

Constants. LSRP uses five constants: r, d_s, d_c, d_{sc}, and d_{syn}. 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; and I_{syn} is used to control the frequency of information update between neighboring nodes.

Variables. As in existing distance-vector routing protocols, each node i maintains the two variables $d.i$ and $p.i$, where $d.i$ records the 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). (Note that, by definition, $d.i$ and $p.i$ are the only problem-specific variables for a node i in LSRP.) To achieve local stabilization, each node i also maintains a boolean variable $ghost.i$. $ghost.i$ is $true$ if node i is being involved in a containment wave.

To enable inter-node coordination, i maintains "mirror" variables $d.i'.i$, $p.i'.i$, and $ghost.i'.i$ for each neighbor i', which denote i's knowledge of the latest values of $d.i'$, $p.i'$, and $ghost.i'$ respectively. (For convenience in presentation, we also regard $d.i.i$ as $d.i$, $p.i.i$ as $p.i$, and $ghost.i.i$ as $ghost.i$.) To control the frequency of information update between neighboring nodes, variable $t.i$ is used to denote the time when i broadcasts the values of $d.i$, $p.i$, and $ghost.i$ last time.

```
Protocol     $LSRP.i$
Constant     $r$ : node-id
             $d_s,\ d_c,\ d_{sc},\ I_{syn}$ : real
Var          $d.i,\ d.i'.i\ (i' \in N.i)$ : integer
             $p.i,\ p.i'.i\ (i' \in N.i)$ : node-id
             $ghost.i,\ ghost.i'.i\ (i' \in N.i)$ : boolean
             $t.i$ : real
             $k$ : node-id
Parameter    $j$ : node-id
Actions
 $\langle S_1 \rangle :: MP.i \land p.i \neq i \longrightarrow$
                $p.i := i;$
                $t.i := CLK.i;$
                send $m(p.i)$ to $N.i$
 []
 $\langle S_2 \rangle :: SW.i.j \land \neg ghost.j.i \xrightarrow{d_s}$
                $d.i, p.i := d.j.i + w.i.j, j;$
                $ghost.i := false;$
                $t.i := CLK.i;$
                send $m(d.i, p.i, ghost.i)$ to $N.i$
 []
 * * *
 $\langle C_1 \rangle :: \neg ghost.i \land (SP.i \lor CW.i) \xrightarrow{d_c}$
                $ghost.i := true;$
                if $SP.i \rightarrow p.i := i$ fi;
                $t.i := CLK.i;$
                send $m(p.i, ghost.i)$ to $N.i$
 []
 $\langle C_2 \rangle :: ghost.i \land$
          $\neg(\exists k : k \in N.i \land p.k.i = i \land d.k.i = d.i + w.i.k) \longrightarrow$
                $ghost.i := false;$
                if $i = r \rightarrow d.i, p.i := 0, i$
                   []
                   $i \neq r \land PS.i.j \rightarrow d.i, p.i := d.j.i + w.i.j, j$
                   []
                   $i \neq r \land \neg(\exists k : PS.i.k) \rightarrow d.i, p.i := \infty, i$
                fi;
                $t.i := CLK.i;$
                send $m(d.i, p.i, ghost.i)$ to $N.i$
 []
 * * *
 $\langle SC \rangle :: ghost.i \land SCW.i \xrightarrow{d_{sc}}$
                $ghost.i := false;$
                if $p.i = i \rightarrow$
                   do $d.k.i + w.i.k = d.i \land SW.i.k \rightarrow$
                        $p.i := k$
                   od;
                fi;
                $t.i := CLK.i;$
                send $m(ghost.i)$ to $N.i$
 []
 * * *
 $\langle SYN_1 \rangle :: (t.i + I_{syn} \leq CLK.i) \lor (t.i > CLK.i) \longrightarrow$
                $t.i := CLK.i;$
                send $m(d.i, p.i, ghost.i)$ to $N.i$
 []
 $\langle SYN_2 \rangle ::$ rcv $m$ from $j \longrightarrow$
                update $d.j.i$, $p.j.i$, and/or $ghost.j.i$ accordingly
```

Fig. 4. LSRP: local stabilization in shortest path routing

For clarity of presentation, we let $N.i$ be the set of neighboring nodes of i, we let $CLK.i$ be the clock value of i, and we let $w.i.i$ be ∞ (i.e., there is no self-loop). A dummy variable k is also used.

Protocol actions. As discussed in Section IV-C, the diffusing computation in LSRP consists of three diffusing waves: the stabilization wave, the containment wave, and the super-containment wave. These diffusing waves are implemented in LSRP as follows:

- Actions S_1 and S_2 implement the stabilization wave. More specifically, S_2 implements the distributed Bellman-Ford algorithm; S_1 guarantees that the next-hop of a node i (i.e., $p.i$) is consistent with the information within its neighborhood (i.e., S_1 is introduced to guarantee self-stabilization).
- Actions C_1 and C_2 implement the containment wave. To enable a super-containment wave to trace a mistakenly initiated containment wave by the parent-child relationship between neighboring nodes, each containment wave is a *round* procedure consisting of a phase of propagating outward and a phase of shrinking back. Action C_1 implements the phase of propagating outward which is to stop the corresponding stabilization wave, and action C_2 implements the phase of shrinking back after the stabilization wave has been stopped.
- Action SC implements the super-containment wave. The super-containment wave propagates along the path signified by the parent-child relationship maintained in the corresponding containment wave.

The propagation speed of a diffusing wave is controlled by the guard hold-time of the actions implementing the wave. To guarantee that containment waves propagate faster than stabilization waves and that super-containment waves propagate faster than containment waves in the presence of clock drift as well as message passing delay, the guard hold-time d_s, d_c, and d_{sc} used in LSRP should be such that $d_s > \alpha \cdot (d_c + U)$, $d_c > \alpha \cdot (d_{sc} + U)$, and $d_{sc} \geq 0$. (For clarity of presentation, we relegate the detailed reasoning to [17].)

To update information between neighboring nodes, actions SYN_1 and SYN_2 are used.

We further elaborate on the protocol actions as follows.

Stabilization wave. By implementing the distributed Bellman-Ford algorithm with some changes to cooperate with containment wave, the stabilization wave guarantees that a system eventually stabilizes to a legitimate state.

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. Then, i sets $t.i$ to its current clock value, and broadcasts the new value of $p.i$ to its neighbors.

$MP.i$ is defined as

$$(i = r \wedge d.i = 0) \vee (ghost.i \wedge SP.i), \text{where } SP.i = true \text{ if } i \text{ is a source of fault propagation.}$$

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.

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 d_s time, then i sets j as its parent, and sets $d.i$, $ghost.i$ to $d.j.i + w.i.j$, $false$ respectively. Also, i sets $t.i$ to its current clock value, and broadcasts the new values of $d.i$, $p.i$, and $ghost.i$ to its neighbors.

$SW.i.j$ is defined as

$$\begin{array}{l} j \in N.i \ \wedge \ d.j.i + w.i.j \leq d.i \ \wedge \\ (\forall k : k \in N.i \Rightarrow d.j.i + w.i.j \leq d.k.i + w.i.k) \ \wedge \\ ((j \neq p.i \wedge (p.i \in N.i \wedge \neg ghost.(p.i).i \Rightarrow \\ \qquad\qquad d.j.i + w.i.j < d.(p.i).i + w.i.(p.i))) \\ \vee \\ (j = p.i \wedge d.i \neq d.j.i + w.i.j)) \end{array}$$

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

- ○ j is the neighbor of i via which the distance value of i is the smallest; and
- ○ if j is not the current parent of i, then the distance value of i via j is less than that via $p.i$ unless $p.i$ is not a neighbor of i or is involved in a containment wave; if j is the current parent of i, then the distance value of i is not equal to that of j plus $w.i.j$.

However, node i should not propagate any stabilization wave from j if j is being involved in a containment wave, since the state of any node being involved in a containment wave is corrupted.

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 d_c time, 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.[4] Then, i sets $t.i$ to its current clock value, and broadcasts the new values of $p.i$ and $ghost.i$ to its neighbors.

$SP.i$ is defined as

$$\begin{array}{l} (\forall j : j \in N.i \wedge \neg ghost.j.i \Rightarrow d.j.i + w.i.j > d.i) \ \wedge \\ ((i \neq r \wedge d.i \neq \infty \wedge d.i \neq d.(p.i).i + w.i.(p.i)) \ \vee \\ \ (i = r \wedge d.i \neq 0)) \end{array}$$

That is, a node i is a source of fault propagation if none of its neighbors that are not involved in any containment wave can offer i a distance value that is no greater than what i currently has, and either i is not the destination node and its distance value is not consistent with that of its parent, or i is the destination node and its distance value is not 0.

$CW.i$ is defined as

$$\begin{array}{l} p.i \in N.i \wedge ghost.(p.i).i \wedge d.i = d.(p.i).i + w.i.(p.i) \ \wedge \\ \neg(\exists k : k \in N.i \wedge \neg ghost.k.i \wedge d.k.i + w.i.k \leq d.i) \end{array}$$

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., $d.i = d.(p.i).i + w.i.(p.i)$), and i does not have a

[4] Conceptually, when i is a source of fault propagation, it is a local minimum in terms of distance values and no neighbor can act as its next-hop by providing a smaller distance to the destination. Therefore, i sets $p.i$ to i. By this design, the destination node r can "stabilize" $p.r$ to r when $d.r \neq 0$; a node i that is in a loop (e.g., due to state corruption) can break the loop within constant time by setting $p.i$ to itself.

neighbor k that is not involved in any containment wave and can offer i a distance value smaller than what i currently has.

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., $p.k.i = i$ and $d.k.i = d.i + w.i.k$), 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 $d.j.i + w.i.j$ 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 to guarantee loop freedom during stabilization.

  $PS.i.j$ is defined as

  $$\begin{aligned} & j \in N.i \wedge \neg ghost.j.i \wedge p.j.i \neq i \ \wedge \\ & d.j.i + w.i.j \leq d.i \ \wedge \\ & (\forall k : k \in N.i \wedge \neg ghost.k.i \Rightarrow \\ & \qquad\qquad d.j.i + w.i.j \leq d.k.i + w.i.k) \end{aligned}$$

  That is, node j is a parent substitute of i if j is not a child of i, and, among all the neighbors of i that are not involved in any containment wave, j offers i the smallest distance value that is no greater than what i currently has.

Also, i sets $t.i$ to its current clock value, and broadcasts the new values of $d.i$, $p.i$, and $ghost.i$ to its neighbors.

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 d_{sc} time, then i sets $ghost.i$ to $false$. Moreover, if i has set $p.i$ to itself because it was a source of fault propagation, i recovers its parent immediately. Then, i sets $t.i$ to its current clock value, and broadcasts the new value of $ghost.i$ to its neighbors.

$SCW.i$ is defined as

$$\begin{aligned} & (i = r \wedge d.i = 0) \ \vee \\ & (i \neq r \wedge \neg SP.i \wedge (p.i \neq i \Rightarrow \neg ghost.(p.i).i)) \end{aligned}$$

That is, 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.

Information update. Actions SYN_1 and SYN_2 guarantee that the values of mirror variables $d.i'.i$, $p.i'.i$, and $ghost.i'.i$ at node i stabilize to those of $d.i'$, $p.i'$, and $ghost.i'$ for every neighbor i' of i.

Action SYN_1: if i did not broadcast the values of $d.i$, $p.i$, and $ghost.i$ in the past I_{syn} time (i.e., $t.i + I_{syn} \leq CLK.i$) or if variable $t.i$ is corrupted to be greater than the current clock value of i, i sets $t.i$ to its current clock value, and broadcasts the values of $d.i$, $p.i$, and $ghost.i$ to its neighbors.

Action SYN_2: when i receives the latest values of $d.j$, $p.j$, and/or $ghost.j$ from its neighbor j, i records the values to variables $d.j.i$, $p.j.i$, and/or $ghost.j.i$ respectively.

E. Examples revisited

To illustrate how LSRP behaves in the presence of faults, we reconsider the examples discussed in previous sections. For simplicity of presentation, we assume that $\alpha = 1$, link delay is a constant u, processing delay is negligible, containment waves propagate twice as fast as stabilization waves (i.e, $d_s = 2d_c + u$), and super-containment waves propagate four times as fast as containment waves (i.e., $d_c = 4d_{sc} + 3u$).

We first study the case where $d.v_8$ is corrupted to 1 and nodes v_6 and v_5 have learned the corrupted value when the system is at the state as shown in Figure 1. The system behavior after the state corruption at v_8 is shown by the space-time diagram in Figure 5:

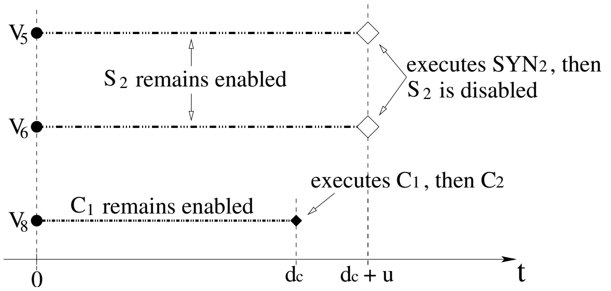


Fig. 5. System behavior after $d.v_8$ is corrupted to 1.

- First, the guard for action C_1 evaluates to true at v_8 (because v_8 is a source of fault propagation, by definition), and the guard for action S_2 evaluates to true at v_6 and v_5. Therefore, C_1 becomes enabled at v_8 and S_2 becomes enabled at v_6 and v_5. For convenience, we regard this moment as time 0.
- C_1 remains enabled at v_8 until time d_c, when v_8 executes C_1. After the execution of C_1, C_2 becomes enabled at v_8 and is executed immediately, since the guard hold-time for C_2 is zero. The execution of C_2 corrects $d.v_8$ to 3.
- C_2 remains enabled at v_6 and v_5 until time $d_c + u$, when v_6 and v_5 receive messages from v_8 reflecting its new state. Therefore, action SYN_2 is executed twice (processing the two messages from v_8) at v_6 and v_5 at time $d_c + u$, which disables S_2 at v_6 and v_5. At this moment, the system reaches a legitimate state.

In the above process, only actions 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 result achievable.

Faults may not be ideally contained in all cases, but they are always contained locally around where they occur in LSRP. To illustrate this, we consider the case where $d.v_{11}$ is corrupted to 2 and v_{13} has learned the corrupted value when the system is at the state as shown in Figure 1. The system behavior after the state corruption at v_{11} is shown in Figure 6:

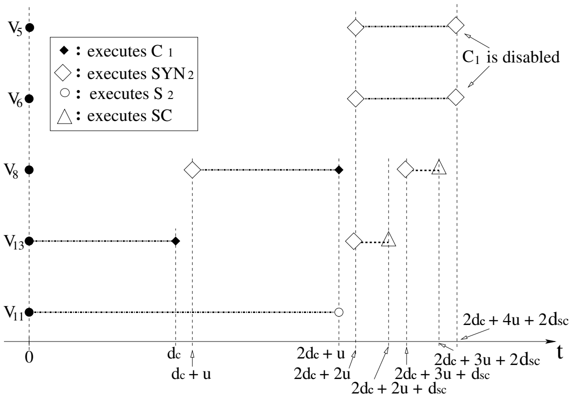


Fig. 6. System behavior after $d.v_{11}$ is corrupted to 2.

- First, action S_2 becomes enabled at v_{11}, and action C_1 becomes enabled at v_{13}. For convenience, we regard this moment as time 0.
- At time d_c, v_{13} executes C_1 and sends its new state to v_8 (by which the containment wave propagates from v_{13} to v_8).
- The new state of v_{13} reaches v_8 at time $d_c + u$, when action SYN_2 is enabled and executed immediately at v_8. As a result, action C_1 becomes enabled at v_8 at time $d_c + u$.
- Then, v_8 executes C_1 and v_{11} executes S_2 at time $2d_c+u$. u time later, v_6 and v_5 receive the new state of v_8, and v_{13} learns the new state of v_{11}. Consequently, C_1 becomes enabled at v_6 and v_5, and SC becomes enabled at v_{13}.
- At time $2d_c + 2u + d_{sc}$, v_{13} executes SC and sends its corrected state to v_8 (by which the super-containment wave propagates from v_{13} to v_8). As a result, SC becomes enabled at v_8 at time $2d_c + 2u + d_{sc}$.
- d_{sc} time later, v_8 executes SC and sends its new state to v_6 and v_5 (by which the super-containment wave propagates to v_6 and v_5).
- At time $2d_c + 4u + 2d_{sc}$, action SYN_2 becomes enabled and is executed immediately at v_6 and v_5, which in turn disables C_1 at v_6 and v_5. As a result, the system reaches its legitimate state.

In the above process, only nodes v_{11}, v_{13}, and v_8 execute one or more protocol actions, while the rest of the system remain unaffected. Therefore, the impact of the state corruption at v_{11} is contained locally (i.e., within 2 hops in the above case).

V. Protocol analysis

In this section, we present 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. (For clarity of presentation, we relegate the proofs of all the theorems and lemmas in the paper to the appendix and [17].)

A. Property of local stabilization

Given a system topology $G'(V', E')$, we define predicate $\mathcal{L}$ as

$$(\forall i : i \in V' \Rightarrow \neg ghost.i \wedge LH.i) \wedge$$
$$(\forall e : e \in E' \Rightarrow \textit{there is no message in } e)$$

where $LH.i$ is defined as

$$(i = r \Rightarrow d.i = 0 \wedge p.i = i) \wedge$$
$$(i \neq r \Rightarrow d.i = d.(p.i) + w.i.(p.i) \wedge p.i \in N.i \wedge$$
$$(\forall k : k \in N.i \Rightarrow d.(p.i) + w.i.(p.i) \leq d.k + w.i.k))$$

Then, every 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 and the next-hop on its shortest path to r. For LSRP, we have

Theorem 1 (Self-stabilization): Starting at an arbitrary state, every computation of a system where LSRP is used is guaranteed to reach a state in $\mathcal{L}$. □

From Theorems 1, we know that LSRP guarantees the formation of the shortest path tree in a system when it starts at an arbitrary state.

Furthermore, local stabilization is guaranteed in LSRP. The analysis is as follows.

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

Lemma 1: 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 $O(P(q_0))$. □

When the set of perturbed nodes are not contiguous, there are multiple perturbed regions (denoted as S_0, S_1, ..., S_m, $m \geq 1$) in the system. For each perturbed region S_i, we define its *containment region* CR_i as 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 are *disjoint* if there do not 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 do not exist any two containment regions CR_i and CR_j ($i \neq j$) that are adjoining. Then, we have

Lemma 2: 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 $O(\max_{i \in 0..m} |S_i|)$. □

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 1: 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 $O(\max_{i \in 0..m} |S_i|)$. □

Multiple containment regions CR_0, CR_1, …, CR_m ($m \geq 1$) are 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}, …, 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. Then, we have

Lemma 3: 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 $O(\max_{i \in 0..m} |S_i|)$. □

Lemmas 2 and 3 imply

Corollary 2: 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 $o(P(q_0))$). □

Lemma 1 and Corollary 2 imply

Theorem 2 (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 $O(\text{MAXP})$, 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 2, we see that LSRP solves the shortest path routing problem in a linear-local stabilizing manner.

B. Properties of loop freedom and quick loop removal

Theorem 3 (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. □

From Theorem 3, 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 4 (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. □

Theorems 3 and 4 imply

Corollary 3: 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 $(d_s + U)$ time. □

VI. Discussion

In this section, we discuss the impact of network topology on local stabilization, and we discuss issues related to the application of LSRP.

A. 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 in the sense that higher edge density is conducive to local stabilization.

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, a mistakenly initiated containment wave, if any, will propagate no farther in $G.q'$ than in $G.q$ 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,

Proposition 1: 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'. □

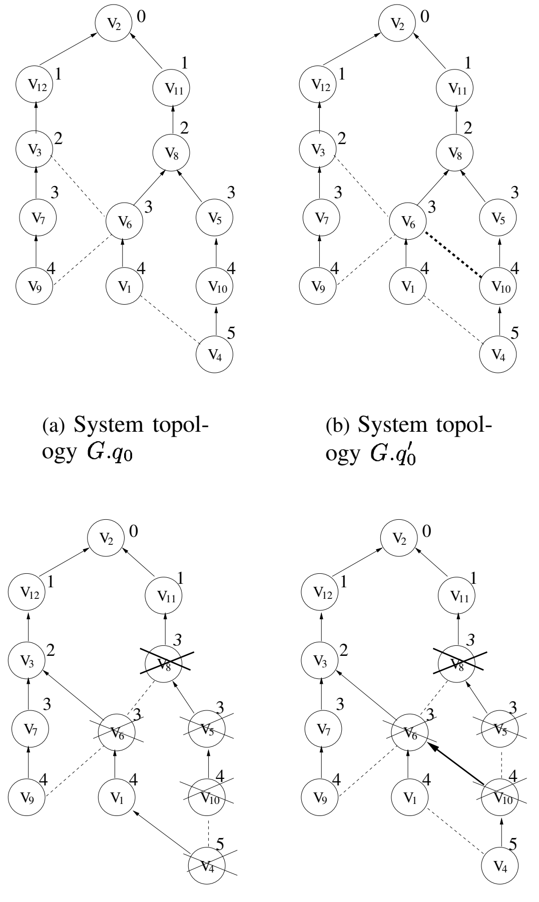
One example is shown in Figure 7. Figure 7(a) and (b) represent system topology $G.q_0$ and $G.q_0'$ at legitimate states q_0 and q_0' respectively. The only difference between $G.q_0$ and $G.q_0'$ is that edge (v_6, v_{10}) is not in $G.q_0$ but is in $G.q_0'$. Then,

- 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 7(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 [4], 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.

B. Issues related to the application of LSRP

Accommodating continuous faults. 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-



(a) System topology $G.q_0$

(b) System topology $G.q_0'$

(c) Perturbed state in $G.q_0$

(d) Perturbed state in $G.q_0'$

Fig. 7. An example where denser edges reduce the perturbation size and range of contamination upon perturbations in the system

domain routing [2]. Due to its property of local stabilization, LSRP also accommodates these kinds of faults, to an extent depending on the perturbation size, while still guaranteeing local fault containment and local stabilization. We elaborate on this as follows.

Theorem 2 implies

Corollary 4: If a 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 $o(P(q_0))$. □

From Corollary 4, we see that when a fault keeps occurring, the impact of the fault is still locally contained as long as the interval between two consecutive occurrences is asymptotically greater than the perturbation size of the fault.

Moreover, we have

Theorem 5: 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 $t_0 + T$ ($T \geq 0$), then every system computation reaches a state in $\mathcal{L}$ within $O(P(q_0)+T)$ time, and the range of contamination is $O(P(q_0) + T)$. □

From Theorem 5, 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.

Speed of diffusing waves. In LSRP, parameters d_s, d_c, and d_{sc} control the propagation speed of stabilization waves, containment waves, and super-containment waves respectively. Therefore, the relationship between these three parameters determines the degree of fault containment in the presence of faults. Especially, we need to choose d_s and d_c carefully: on one hand, the larger the ratio $\frac{d_s}{d_c}$ is, the more tightly a mistakenly initiated stabilization wave is contained; on the other hand, the smaller the ratio $\frac{d_s}{d_c}$ is, the more tightly a mistakenly initiated containment wave tends to be contained (since the corresponding super-containment wave will not be initiated until certain stabilization wave is executed, as shown in Figure 6). In practice, we should consider the probabilities of stabilization or containment waves being mistakenly initiated and the degree of fault containment we expect in choosing the parameters.

Control overhead. The impact of faults is locally contained in LSRP, therefore the control overhead (e.g., the number of control messages) is also a function of the perturbation size, instead of the system size. Consequently, LSRP asymptotically reduces the control overhead when compared with non-locally-stabilizing protocols such as DUAL and LPA.

Moreover, the diffusing computation involved in stabilization and containment waves of LSRP also (implicitly) exists in other routing protocols (such as DUAL and LPA) to guarantee convergence as well as loop freedom. Therefore, stabilization and containment waves in LSRP do not introduce much more overhead except for the bit used to encode the variable $ghost$. Of course, the overhead associated with super-containment waves exists only in LSRP and not in other protocols; but this overhead is low because it only needs one bit to encode the variable $ghost$, and it is local in the sense that it is bounded from above by a function of the perturbation size.

VII. Concluding remarks

To formally characterize properties of local stabilization in networked and distributed systems, we formulated the concepts of perturbation size, $\mathcal{F}$-local stabilization, and range of contamination. 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. LSRP guarantees both local stabilization and loop freedom during stabilization. In LSRP, we 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, timer MinRouteAdvertisementInterval is used in BGP to control the frequency of route exchange between BGP peers. The timer is similar to the delay introduced for the stabilization wave in LSRP. In implementing LSRP, we only need to introduce smaller timers for the containment wave and super-containment wave.

We 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 the interesting question of 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 [5], formation of routing loops is regarded as problematic, and a variety of schemes have been proposed to avoid forming loops, such as those used in EIGRP, OSPF, and BGP. 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 tolerate such faults as misconfiguration and persistent congestion, which are special cases of state corruption. In 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.

Acknowledgment

We thank Mohamed Gouda, Ted Herman, and Shay Kutten for their helpful comments. We also thank the anonymous referees, whose comments have significantly improved the quality of this paper.

References

[1] C. Labovitz, G. R. Malan, and F. Jahanian, "Origins of internet routing instability," in *IEEE INFOCOM*, 1999, pp. 218–226.

[2] 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, pp. 138–147.

[3] M. Nesterenko and A. Arora, "Local tolerance to unbounded byzantine faults," in *IEEE SRDS*, 2002, pp. 22–31.

[4] H. Zhang and A. Arora, "GS^3: Scalable self-configuration and self-healing in wireless sensor networks," *Computer Networks (Elsevier)*, vol. 43, no. 4, pp. 459–480, 2003.

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

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

[7] C. Labovitz, A. Ahuja, R. Wattenhofer, and S. Venkatachary, "The impact of internet policy and topology on delayed routing convergence," in *IEEE INFOCOM*, 2001, pp. 537–546.

[8] A. Shaikh, L. Kalampoukas, R. Dube, and A. Varma, "Routing stability in congested networks: Experimentation and analysis," in *ACM SIGCOMM*, 2000, pp. 163–174.

[9] M. Jayaram and G. Varghese, "Crash failures can drive protocols to arbitrary states," in *ACM PODC*, 1996, pp. 247 – 256.

[10] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju, "Fault-containing self-stabilizing algorithms," in *ACM PODC*, 1996, pp. 45–54.

[11] Y. Azar, S. Kutten, and B. Patt-Shamir, "Distributed error confinement," in *ACM PODC*, 2003, pp. 33–42.

[12] S. Ghosh and X. He, "Scalable self-stabilization," in *IEEE ICDCS'WSS*, 1999, pp. 18–24.

[13] J. J. Garcia-Lunes-Aceves, "Loop-free routing using diffusing computations," *IEEE/ACM Transactions on Networking*, vol. 1, no. 1, pp. 130–141, 1993.

[14] J. J. Garcia-Lunes-Aceves and S. Murthy, "A path-finding algorithm for loop-free routing," *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 148–160, 1997.

[15] M. G. Gouda, *Elements of Network Protocol Design*. John Wiley and Sons, 1998.

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

[17] A. Arora and H. Zhang, "Local stabilization in shortest path routing," *Technical Report, OSU-CISRC-7/03-TR45, The Ohio State University (ftp://ftp.cis.ohio-state.edu/pub/tech-report/2003/TR45.ps)*, July, 2003.

Appendix

In this appendix, we present the proof sketch for Lemma 1. Due to limitation of space, the complete proofs of all the theorems and lemmas in the paper are relegated to [17].

Lemma 1: 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.[5] The set of nodes that are not in R_p is called the *healthy region* R_h[6]. A node i in R_p is called a *perturbed boundary node* if it has a neighbor j that is in R_h, and node j is called a *healthy boundary node* correspondingly. Given R_p and R_h at q_0, we let $P_d = max\{hops(i, R_h) : i \in R_p\}$, where $hops(i, R_h) = min\{hops(i, j, G') : j \in R_h\}$ and $hops(i, j, G')$ = the number of hops in the shortest path between nodes i and j in G'.

There are three major 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. (Moreover, the effect of the corruption of mirror variables is the same as the corruption of $ghost.i$, except that the former adds constant factor I_{syn} to the asymptotic expression of the theorems in the paper. Thus, for simplicity of presentation, we skip it here.)

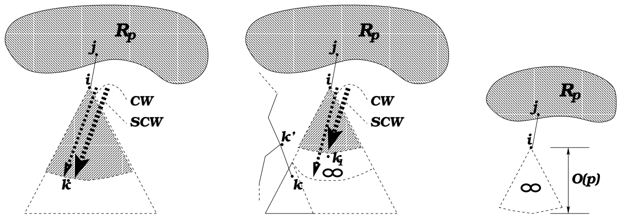
When a single perturbed region occurs in the system, there are two cases: (i) the "distance value" of every perturbed node is no less than what all the healthy boundary nodes can offer, i.e., ($\forall i, j : i \in R_p \wedge j$ is a healthy boundary node $\Rightarrow d.i \geq d.j + \min\{w.j.k : k \in R_p \wedge (j, k) \in E'\}$); (ii) the "distance value" of some or all perturbed nodes is less than what some or all healthy boundary nodes can offer. Due to limitation of space, here we only present the proof sketch for case (i). (The proof method for case (ii) is similar, and the proof is relegated to [17].)

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

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

For case (i), here we only consider the subcase 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. Due to limitation of space, we only analyze the latter case here.

When there are some non-root 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. Also, here we only 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 8). Then, the parent of 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 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)

Fig. 8. The "distance value" of every perturbed node is no less than what all the healthy boundary nodes can offer, and there are some sources of fault propagation. In the figure, SW, CW, and SCW stand for stabilization wave, containment wave, and super-containment wave respectively.

(I) If action C_2 has not been executed at node i before the moment that is one 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 8(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.

We first show that starting at state q_0, the system reaches a state $q_0'$where $(\forall k : k \in V' \Rightarrow \neg ghost.i \wedge \neg SP.k)$ 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 1 too. As shown in the proof for Theorem 1, 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 $(\forall k : k \in V' \Rightarrow \neg ghost.i \wedge \neg SP.k)$ holds.

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 8(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, $d_c > \alpha \cdot (d_{sc} + U)$ 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

$p_{cw} = \frac{O(p)}{d_c - d_{sc} + L - U} = O(p)$.
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 8(b)).

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

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

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

  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 8(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 $O(p)$.

□