LSRP: Local Stabilization in Shortest Path Routing *

Anish Arora Hongwei Zhang
Department of Computer and Information Science
The Ohio State University, USA
{anish, zhangho}@cis.ohio-state.edu

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 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 mistakenly initiated stabilization wave, the super-containment wave contains 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.

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 [9, 14].

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 [11, 14, 15], 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 protocols.

Distance-vector (and its variant, path-vector) protocols for the Internet, such as Routing Information Protocol (RIP) and BGP, have long been studied [6]. 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 [12]. 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 [14], and result in severe instability [8, 14]. 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 [9, 13, 14]. And theoretically speaking, even simple faults such as node crash and message loss, can drive a network into arbitrary states [7]. 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

*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.

of contamination, in order to characterize the local stabilization properties of systems. These concepts are generically applicable to distributed computing and networking problems.

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 is scalable in the sense that each node only communicates with and maintains information about its 1-hop neighbors.

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 in Section 7 and make concluding remarks in Section 8.

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. The ratio of clock rates 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.

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

[1] We choose a shared memory model only for simplicity of presentation. Our protocol LSRP is readily adaptable to the message passing model (see [16]).

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 $e \in [l.a, u.a]$ such that 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.

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.

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:

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

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

Then, perturbation size is defined as follows.

[2]The node set V' and edge set E' should be such that $G_-(V_-, E_-)$ and $G_+(V_+, E_+)$ are valid graphs.

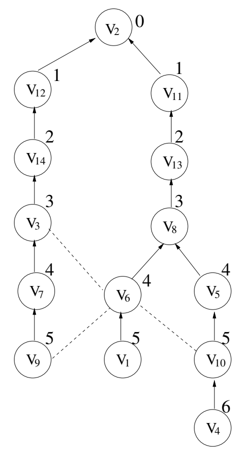
Definition 1 (Perturbation size) *The perturbation size at a system state q, denoted as $P(q)$, is $\min_{q' \in Q_l} |A_{q'} \cup B_{q'}|$ where Q_l is the set of all possible legitimate system states,*

$$A_{q'} = \{i : i \in (V.q \cap V.q') \wedge q(i) \neq q'(i)\},$$
$$B_{q'} = \{i : i \in V.q \wedge i \in (DS_{q'}(V.q' \setminus V.q, E.q' \setminus E.q) \cup DS_{q'}(V.q \setminus V.q', E.q \setminus E.q'))\}$$

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.

Then, the *set of potentially perturbed node sets at state q*, denoted as $PP(q)$, is 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 the system



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

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 node sets 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 node sets 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 and $\mathcal{F}$ is a linear function, we say that a system is 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 *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 there is at least one action executed at the node during stabilization. Then, the *range of contamination*, denoted by $R_c(q)$, is defined as the the maximum distance from the set of contaminated nodes to the perturbed node set $PN(q)$.

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.
- 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.
- 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.

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 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.

4.2 Fault propagation in existing distance-vector protocols

Existing distance-vector routing protocols are based on the distributed Bellman-Ford algorithm [6, 10]. 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 $d.j + w_{ij}$, 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

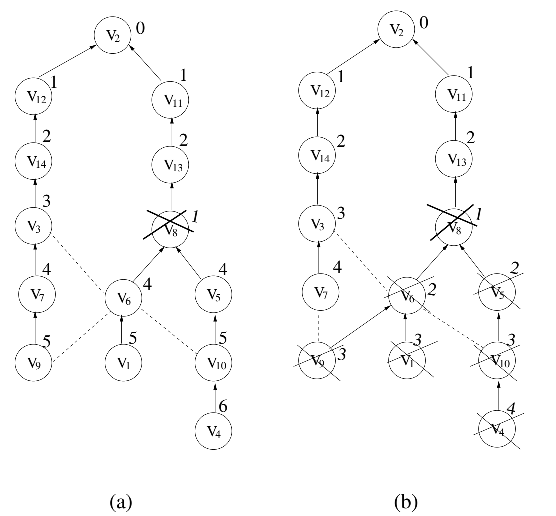


Figure 2. Example of fault propagation in existing distance-vector routing protocols

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.

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. 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.

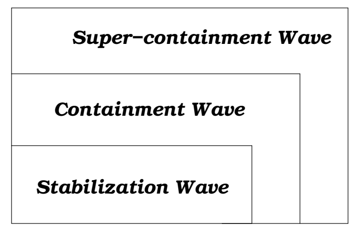


Figure 3. Layering of diffusing waves in shortest path routing

The stabilization wave is a diffusing computation that implements the basic distributed Bellman-Ford algorithm with some changes in order to cooperate with the containment wave. A stabilization wave can propagate the "correction" action that makes a system 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 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 either the node is not the destination node, or is its distance value is not 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. 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.

For the above wave-layering approach to work, the super-containment wave must self-stabilize itself locally upon perturbations, 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 [6] 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.

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 [16].)

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. (For simplicity, we relegate the detailed reasoning to [16].)

Protocol $LSRP.i$
Constant r : node_id
$d_s,\ d_c,\ d_{sc},\ L,\ U$: real
Var $d.i$: integer
$p.i$: node_id
$ghost.i$: boolean
k : node_id
Parameter j : node_id
Actions

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

[]

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

[]

* * *

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

[]

$\langle C_2 \rangle :: ghost.i \wedge \neg(\exists k : k \in N.i \wedge p.k = i \wedge d.k = d.i + 1) \longrightarrow$
$ghost.i := false;$
if $i = r \rightarrow d.i, p.i := 0, i$
[]
$i \neq r \wedge PS.i.j \rightarrow d.i, p.i := d.j + 1, j$
[]
$i \neq r \wedge \neg(\exists k : PS.i.k) \rightarrow d.i, p.i := \infty, i$
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

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 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

$(i = r \wedge d.i = 0) \vee (ghost.i \wedge SP.i)$, where $SP.i = true$ if i 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.

$\underline{Action\ S_2}$: 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.

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

$\underline{Action\ C_1}$: 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 either it is 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.

$\underline{Action\ C_2}$: 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.

$\underline{Action\ SC}$: 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

We reconsider the example 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.

5 Protocol analysis

In this section, we present the fixpoint[3] 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. (The proofs of all the theorems and lemmas are relegated to [16].)

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

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

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. Therefore, LSRP guarantees the formation of the shortest path tree in a system.

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

[3]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.

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 $S_0, S_1, \ldots, S_m$, $m \geq 1$) 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 $CR_0, CR_1, \ldots, CR_m$ ($m \geq 1$) 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 $CR_{k_0}, CR_{k_1}, \ldots, CR_{k_t}$ 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. 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 $O(\max_{i \in 0..m} |S_i|)$.*

Lemma 1, 2, and 3 imply

Theorem 1 (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 1, we see that LSRP solves the shortest path routing problem in a linear-local stabilizing manner.

5.2 Properties of loop freedom and quick loop removal

Theorem 2 (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 2, 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 3 (Quick loop removal) *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 $(d_{sc} + U)$ time.*

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 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, the potential mistakenly initiated containment wave will propagate no farther in $G.q'$ than in $G.q$ (see [16] for details). 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 1.

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'.*

(We give an example where higher edge density helps in local stabilization in [16].)

In wireless networks, especially in wireless sensor networks [15], 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

A loop-free distance-vector protocol DUAL was proposed in [3] and incorporated into EIGRP [6]. 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 [14]. Furthermore, in DUAL, if a loop is already formed (e.g., due to state corruption), the time taken to break the loop is proportional to the length of the loop; in contrast, a loop can be broken within small constant time in LSRP, irrespective of the loop length.

In [15], 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. In [4], 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 [5], 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 [4] and [11], the concept of fault containment was proposed. However, in [4], 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 [11], 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[4] is not constant.

[4]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.

In [2], a self-stabilizing algorithm was proposed for maintaining a spanning tree in a network. Time-optimal versions were proposed later [1]. But none of these contain faults locally or achieve local stabilization.

8 Concluding remarks

We 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. 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 [6], 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 congestions, 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.

We 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) routing protocols, such as DSDV, AODV, RIP, and BGP.

Acknowledgment

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

References

[1] S. Aggarwal and S. Kutten. Time optimal self-stabilizing spanning tree algorithms. In *FSTTCS*, pages 400–410, 1993.

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

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

[4] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *ACM PODC*, pages 45–54, 1996.

[5] S. Ghosh and X. He. Scalable self-stabilization. In *IEEE ICDCS'WSS*, pages 18–24, 1999.

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

[7] M. Jayaram and G. Varghese. Crash failures can drive protocols to arbitrary states. In *ACM PODC*, pages 247–256, 1996.

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

[9] C. Labovitz, G. R. Malan, and F. Jahanian. Origins of internet routing instability. In *IEEE INFOCOM*, pages 218–226, 1999.

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

[11] M. Nesterenko and A. Arora. Local tolerance to unbounded byzantine faults. In *IEEE SRDS*, pages 22–31, 2002.

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

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

[14] 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*, pages 138–147, 2002.

[15] H. Zhang and A. Arora. GS^3: Scalable self-configuration and self-healing in wireless networks. In *ACM PODC*, pages 58–67, 2002.

[16] H. Zhang and A. Arora. LSRP: Local stabilization in shortest path routing. *Technical Report, OSU-CISRC-12/02-TR27, The Ohio State University (ftp://ftp.cis.ohio-state.edu/pub/tech-report/2002/TR27.ps)*, December 2002.