An efficient fault-containing self-stabilizing algorithm for the shortest path problem[*]

Tetz C. Huang
Department of Computer Engineering and Science, Yuan-Ze University,
135 Yuan-Tung Road, Chung-Li 320, Taiwan, R.O.C.
e-mail: cstetz@saturn.yzu.edu.tw

October 31, 2005

Abstract

Shortest path finding has a variety of applications in transportation and communication. In this paper, we propose a fault-containing self-stabilizing algorithm for the shortest path problem in a distributed system. The proposed algorithm has made a considerable improvement in the worst-case stabilization time for single-fault situations, which clearly demonstrates the desirability of a well-designed fault-containing self-stabilizing algorithm. For single-fault situations, the worst-case stabilization time of the proposed algorithm is only $O(\Delta)$, where Δ is the maximum node degree in the system, and the contamination number of the proposed algorithm is 1.

Key Words: Self-stabilizing algorithm - Fault-containment - Single-fault situation - Stabilization time - Shortest path problem

1 Introduction

The notion of self-stabilization in a distributed system was first introduced by Dijkstra [2] in 1974 (cf. also [3, 4]). Adopting Dijkstra's computational model and his definition of self-stabilizing algorithms, Ghosh *et al.* introduced fault-containment of self-stabilizing algorithms in 1996∼1997 [6, 8, 9, 10] (cf. also Gupta [11]). The main idea of fault-containment of self-stabilizing algorithms is to modify an existing self-stabilizing algorithm so that the modified version has the capability of keeping faults from spreading in the system and, more importantly, has the capability of self-stabilizing in a much shorter time, whenever the system incurs only a limited number of transient faults. The motivation to acquire a fault-containing version from an existing self-stabilizing algorithm is also well explained in [10]: " Improved reliability of system components implies that it is more likely that in a well-designed network, the number of components of the system that exhibit transient failures at one time will be small. Hence it is desirable that from such limited transient faults, the system should recover quickly, and in particular, significantly faster than the recovery from failures in an arbitrarily large number of components. Moreover, to ensure that the fault is masked from any higher level application program in most parts of the network, it is important that only a small part of the network around the faulty components make any state changes that can affect the correct operation of a higher level application program". Since the fault-containment for multiple-fault situations is much more difficult to handle, all the above works by Ghosh *et al.* are concentrated on the fault-containment for single-fault situations only. Lin and Huang [17] recently proposed a fault-containing self-stabilizing algorithm for finding a maximal independent set. Although their approach followed closely the spirit of Ghosh *et al.*, they innovated a way of analyzing the "problem-dependent information"(the term used in [10]) for single-fault situations. Mainly due to this analysis of single-fault situations, the faulty node and those nodes around the faulty node can grasp more firmly their neighbors' conditions and, as a consequence, the proposed fault-containing algorithm has a respectable performance in terms of the worst-case stabilization time for single-fault situations. The above is a short review of fault-containment of self-stabilizing algorithms. For the detailed descriptions of fault-containment, readers are referred to the above-mentioned references, in particular [10] and [17] (due to their relevance to this paper).

Self-stabilizing algorithms for finding the shortest paths in a distributed system have been investigated during the past [1, 10, 11, 12, 13, 14, 15, 16]. In [12], Huang and Chen proposed a BFS-tree-finding algorithm that was self-stabilizing under Dijkstra's central demon model. (Note that the BFS tree problem is a special case of the shortest path problem.) Dolev *et al.* [5] first introduced a computational model in which the read/write atomicity was assumed. Under this different computational model, Dolev *et al.* also proposed a self-stabilizing BFS-tree-finding algorithm. A self-stabilizing algorithm for the shortest path problem was proposed in Chandrasekar and Srimani [1] in which the central demon was assumed. Ghosh *et al.* introduced fault-containment of self-stabilizing algorithms and they proposed a fault-

[*]This research was supported in part by the National Science Council, Taipei, Taiwan, under Grant NSC 93-2213-E-155-045.

containing self-stabilizing algorithm for finding a BFS tree in [10]. Their algorithm was based on the algorithm in [12]. Inspired also by [12], Huang and Lin [15], using the bounded function technique, proved that the algorithm in [1] was self-stabilizing under the central demon. In [13] and [16], Huang *et al.* generalized the results in [1] and [15] and showed that the algorithm in [1] and [15] was self-stabilizing under the distributed demon model. In [14], Huang re-examined the Dolev model and proposed a self-stabilizing algorithm for the shortest path problem that generalized the BFS-tree-finding algorithm in [5].

In this paper, we will propose a fault-containing self-stabilizing algorithm for the shortest path problem in a distributed system. Our algorithm is a modification based on the algorithm in [1] and [15]. Although the approach taken is the same as that in [17], the crucial steps, the analysis of single-fault situations and the modification of the original algorithm, are definitely problem-dependent. It is these crucial steps that bring about the good performance of this modified algorithm. For single-fault situations, the worst-case stabilization time of the original algorithm in [1] and [15] is $\Omega(n^k)$, where n is the number of nodes and k is any arbitrary positive integer, whereas the worst-case stabilization time of our fault-containing algorithm is only $O(\Delta)$, where Δ is the maximum node degree.

The rest of the paper is organized as follows: In Section 2, the shortest-path-finding algorithm in [1] and [15] is recalled and the extreme inefficiency of it is demonstrated. In Section 3, the whole process of designing our fault-containing algorithm is exhibited, and the main results concerning the correctness and the efficiency of our algorithm are outlined. An example which illustrates the execution of Algorithm 2 is given in Section 4. Finally in Section 5, some remarks are made; and for the comparison between our algorithm and the algorithm in [10], the computation of the worst-case stabilization time for single-fault situations for the algorithm in [10] is conducted.

2 The algorithm in [1] and [15]

Let $G = (V, E)$ be a simple connected undirected graph that models a distributed system, with each node $i \in V$ representing a processor in the system and each edge $\{i, j\}$ representing the bidirectional link connecting processors i and j. In the system, each edge $e = \{i, j\}$ is preassigned a *weight* $w(e) = w(i, j)$, which is a positive integer. If $L = (e_1, e_2, \ldots, e_t)$ is a path in G, the *weight* (or *length*) of L, $w(L)$, is defined to be $\sum_{k=1}^{t} w(e_k)$. For any two nodes i and j in V, a shortest path between i and j is a path of minimum weight that connects i and j; the weight of a shortest path between i and j is called the *distance between* i *and* j and is denoted by $d(i, j)$. The single-source shortest path problem can be phrased as follows: Suppose a node r in G is specified as the source of the system. We want to find for each node i in G a shortest path between i and the source r. The self-stabilizing shortest-path-finding algorithm in [1] and [15] is as follows. Note that in the algorithm, $d.i$ stands for a local variable of node i and $N(i) = \{j \in V | \{i, j\} \in E\}$ denotes the set of all neighbors of i. The value of each local variable $d.i$ is a non-negative integer. The system assumes Dijkstra's central demon model.

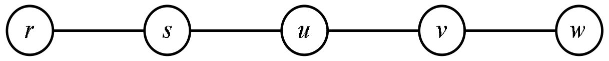

Figure 1: A system of five nodes that has a linear chain topology and is equipped with the algorithm in [1] and [15]

Table 1: An execution of the system in Figure 1 that starts with a single-fault state and ends with a legitimate state

Configuration number	$d.r$	$d.s$	$d.u$	$d.v$	$d.w$
1	25	1	2	3	4
2	25	3	2	3	4
3	25	3	4	3	4
4	25	3	4	5	4
5	25	3	4	5	6
6	25	5	4	5	6
7	25	5	6	5	6
8	25	5	6	7	6
9	25	5	6	7	8
10	25	7	6	7	8
11	25	7	8	7	8
12	25	7	8	9	8
13	25	7	8	9	10
14	25	9	8	9	10
15	25	9	10	9	10
16	25	9	10	11	10
17	25	9	10	11	12
18	25	11	10	11	12
19	25	11	12	11	12
20	25	11	12	13	12
21	25	11	12	13	14
22	25	13	12	13	14
23	25	13	14	13	14
24	25	13	14	15	14
25	25	13	14	15	16
26	25	15	14	15	16
27	25	15	16	15	16
28	25	15	16	17	16
29	25	15	16	17	18
30	25	17	16	17	18
31	25	17	18	17	18
32	25	17	18	19	18
33	25	17	18	19	20
34	25	19	18	19	20
35	25	19	20	19	20
36	25	19	20	21	20
37	25	19	20	21	22
38	25	21	20	21	22
39	25	21	22	21	22
40	25	21	22	23	22
41	25	21	22	23	24
42	25	23	22	23	24
43	25	23	24	23	24
44	25	23	24	25	24
45	25	23	24	25	26
46	25	25	24	25	26
47	25	25	26	25	26
48	25	25	26	27	26
49	25	25	26	27	28
50	25	26	26	27	28
51	25	26	27	27	28
52	25	26	27	28	28
53	25	26	27	28	29
54	0	26	27	28	29
55	0	1	27	28	29
56	0	1	2	28	29
57	0	1	2	3	29
58	0	1	2	3	4

{For the source r}
$R0 : d.r \neq 0 \rightarrow d.r := 0$

{For node $i \neq r$}
$R1 : d.i \neq \min_{j \in N(i)} [d.j + w(i, j)]$
$\rightarrow d.i := \min_{j \in N(i)} [d.j + w(i, j)]$

Legitimate states are defined to be all those global states in which the following condition holds:

$$d.r = 0 \text{ and } \forall i \neq r,\ d.i = \min_{j \in N(i)} [d.j + w(i, j)].$$

There is actually a unique legitimate state and whenever the system reaches the legitimate state, the variable $d.i$ records the distance $d(i, r)$ between i and r for any $i \in V$ (see Lemmas 2 and 3 in [13]). Consequently, finding shortest paths becomes an easy task in the legitimate state (see Concluding Remarks in [15]).

The system in Figure 1 is a linear chain of 5 nodes. We equip it with the above algorithm. Table 1 exhibits an execution of the system that starts with a single-fault

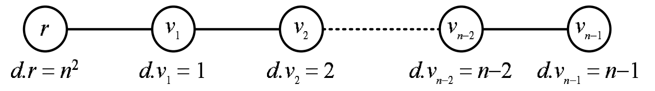

Figure 2: A single-fault state for a linear chain of n nodes

state and ends with a legitimate state. In the table, the shaded area in each configuration indicates the move of the node selected by the central demon. The number of steps of the execution is $(5-1)(\lceil \frac{5^2}{2} \rceil + 1) + 1 = 57$, where $\lceil \frac{5^2}{2} \rceil = 13$ is the ceiling of $\frac{5^2}{2}$. If we generalize the execution in Table 1 in a system whose topology is a linear chain of n nodes, then the initial state of the generalized execution is the single-fault state in Figure 2 and it is not difficult to compute the number of steps of the generalized execution to be $(n-1)(\lceil \frac{n^2}{2} \rceil + 1) + 1$. As a matter of fact, if the local state of node r in Figure 2 is replaced by n^{k-1}, where k is an arbitrary positive integer, then we get a single-fault state for the linear chain of n nodes. Starting with that single-fault state, we can obtain without difficulty an execution that ends with a legitimate state and has its number of steps equal to $(n-1)(\lceil \frac{n^{k-1}}{2} \rceil + 1) + 1$. With the above analysis, we have shown that the worst-case stabilization time of the above algorithm for only single-fault situations is $\Omega(n^k)$, where k is an arbitrary positive integer, that is, the above self-stabilizing shortest-path-finding algorithm is extremely inefficient.

3 Our fault-containing algorithm

Before proceeding to explain the design of our fault-containing self-stabilizing algorithm for solving the single-source shortest path problem, we first introduce two additional variables c and p into each node of the above system in [1] and [15]. The purpose of introducing c and p is to help reveal information about the d-values of each node's neighbors. To be more explicit, we introduce in the following a modified version for the algorithm in [1] and [15]. For ease of presentation, we use $G(i)$ to denote the predicate $d.i \neq \min\limits_{x \in N(i)} [d.x + w(i,x)]$, $A(i)$ to denote the action $d.i := \min\limits_{x \in N(i)} [d.x + w(i,x)]$ and $S(i)$ to denote the set $\{j \in N(i) | d.j + w(i,j) = \min\limits_{x \in N(i)} [d.x + w(i,x)]\}$.

Algorithm 1

```
{For the source $r$}
    $R0 : d.r \neq 0 \rightarrow d.r := 0$

{For node $i \neq r$}
    $R1 : G(i) \rightarrow A(i)$
    $R2 : \neg G(i) \wedge c.i \neq |S(i)| \rightarrow c.i := |S(i)|$
    $R3 : \neg G(i) \wedge c.i = |S(i)| = 1 \wedge p.i \neq j$, where $j$ is the
        unique node in $S(i) \rightarrow p.i := j$
    $R4 : \neg G(i) \wedge c.i = |S(i)| > 1 \wedge p.i \neq \perp \rightarrow p.i := \perp$
```

Note that in the above algorithm, $c.i$ takes values in the set of all positive integers Z^+, $p.i$ takes values in $N(i) \cup \{\perp\}$, and $|S(i)|$ denotes the number of elements in $S(i)$. We claim in the next theorem that the system equipped with the above algorithm is self-stabilizing (although it is still not fault-containing) with the legitimate states being all those states in which the following condition holds:

$d.r = 0 \wedge \forall i \neq r, \neg G(i) \wedge c.i = |S(i)| \wedge [(|S(i)| = 1 \wedge p.i = j$, where j is the unique node in $S(i)) \vee (|S(i)| > 1 \wedge p.i = \perp)]$.

One can easily deduce that there is a unique legitimate state here and the system reaches the legitimate state if and only if the algorithm in the system stops (cf. Lemmas 2 and 3 in [13], and Lemmas 1 and 2 in [15]). One can also see that in a legitimate state, not only is the d-value of each node i of the system equal to the distance $d(i,r)$ between i and r, but also the p-value and c-value of each node i of the system can reveal information about the d-values of i's neighbors. For example, in a legitimate state, the condition $[c.i = 1 \wedge p.i = j]$ reveals that $d.j + w(i,j) = \min\limits_{x \in N(i)} [d.x + w(i,x)]$ and $\forall k \in N(i) - \{j\}$, $d.k + w(i,k) > \min\limits_{x \in N(i)} [d.x + w(i,x)]$.

Lemma 1 (Self-stabilization) *Starting with any initial state, the system will eventually stop in the legitimate state.*

Proof. We first show that the system will stop. Suppose not. Then, there exists an infinite computation $C = (S_1, S_2, ...)$ in the system. If there are infinitely many moves in C which execute $R0$ or $R1$, let S_{i_1}, S_{i_2}, ..., where $i_1 < i_2 <$..., be all the states which result from these moves. Then, if we ignore all the c-values and p-values in the states, then $(S_1, S_{i_1}, S_{i_2}, ...)$ becomes an infinite computation with respect to the algorithm in [15], which is impossible. Hence, there are only finitely many moves in C which execute $R0$ or $R1$. Therefore, there exists a positive integer m such that in the suffix $C' = (S_m, S_{m+1}, ...)$ of C there is no move that executes $R0$ or $R1$. So in C', the d-values of all nodes never change. Thus, the set S of every node i never changes. Hence, every node executes $R2$ at most once in C'. It follows that the system executes $R2$ only finitely many times in C'. So, there exists an integer $l > m$ such that in the suffix $C'' = (S_l, S_{l+1}, ...)$ of C', there is no move that executes $R0$, $R1$ or $R2$. So in C'', d-value, c-value, G-value and set S of every node never change. Thus, every node executes $R3$ or $R4$ at most once, respectively, in C''. It follows that the system executes $R3$ or $R4$ only finitely many times in C''. Therefore, C'' is finite, which causes a contradiction. Hence, we have shown that the system will stop, that is, the system will be in deadlock. However, the deadlock situation can only occur in a legitimate state. Therefore, the lemma is proven. ■

Thus, we have shown that Algorithm 1 is a self-stabilizing algorithm that can solve the single-source shortest path problem for any graph. Of course, Algorithm 1 is still not fault-containing and we will modify it to get a fault-containing algorithm. To enable us to do so, we first make some observations about Algorithm 1 and obtain the following four lemmas.

Lemma 2 *If the d-values of the whole system are at normal, i.e., if $d.r = 0$ and $\forall i \in V - \{r\}, \neg G(i)$, then $\forall i \in V$, $\forall j \in N(i)$, $|d.i - d.j| \leq w(i,j)$.*

Proof. If $u \neq r$ and $v \in N(u)$, then since $d.u = \min_{x \in N(u)} [d.x + w(u,x)]$, $d.u \leq d.v + w(u,v)$. Now let $i \in V$ and $j \in N(i)$.

1) If $i \neq r$ and $j \neq r$, then $d.i \leq d.j + w(i,j)$ and $d.j \leq d.i + w(j,i)$. Hence, $|d.i - d.j| \leq w(i,j)$.

2) If $i = r$, then $d.i = 0$ *and* $j \neq r$. Hence, $d.j \leq d.i + w(j,i) = w(i,j)$. Therefore, $|d.i - d.j| = |-d.j| = d.j \leq w(i,j)$.

3) If $j = r$, then, by the same argument as in 2) above, we have $|d.i - d.j| \leq w(i,j)$. ■

In the following three lemmas, it is assumed that the system starts in a legitimate state and incurs a single fault at a certain time instant t_0 and the single fault causes the change of d-value of the faulty node and possibly its p-value and/or c-value too. Some expressions will be needed in the following proofs as well as in the rest of the paper. So we define them here. For any time instant t, we say that a predicate is true at t^+ if it is true right after t, and that a predicate is true at t^- if it is true right before t. Also, by definition, a predicate is true at t if and only if it is true at both t^+ and t^-; and a predicate is true in a time interval I if it is true at any instant t in the interval. To illustrate how to use these expressions, for instance, if a processor i makes a move to change its d-value from 5 to 3 at a time instant t, then $d.i = 5$ at t^-, $d.i = 3$ at t^+ and $d.i$ at time t is not well-defined. If processor i does not change its d-value at t, then $d.i$ at t^-, $d.i$ at t^+ and $d.i$ at t are all the same.

Lemma 3 *If $i \in V$, $j \in N(i)$ and $|d.i - d.j| > w(i,j)$ at t_0^+, then either i or j is the faulty node.*

Proof. *Since $d.r = 0$ and $\forall i \in V - \{r\}, \neg G(i)$ at t_0^-, we have $|d.i - d.j| \leq w(i,j)$ at t_0^- by Lemma 2. If both i and j are not the faulty node, then $|d.i - d.j| \leq w(i,j)$ at t_0^+, which causes a contradiction. Therefore, either i or j is the faulty node.* ■

Lemma 4 *If $i \neq r$ and i is the faulty node, then $G(i)$ at t_0^+.*

Proof. Since $d.i$ changes and for every $j \in N(i)$, $d.j$ does not change at t_0^+, we have $d.i \neq \min_{x \in N(i)} [d.x + w(x,i)]$ at t_0^+. That is, $G(i)$ at t_0^+. ■

Lemma 5 *If $i \neq r$ and $G(i)$ at t_0^+, then the faulty node must be node i or one of i's neighbors.*

Proof. Suppose neither node i nor any neighbor of node i is the faulty node. Then $d.i$ and $d.j$, for every $j \in N(i)$, do not change at t_0^+. Hence, $d.i = \min_{x \in N(i)} [d.x + w(i,x)]$ at t_0^+, that is, $\neg G(i)$ at t_0^+, which causes a contradiction. Therefore, the faulty node must be node i or one of i's neighbors. ■

Based on the above observations, we now investigate all nontrivial single-fault situations of the system equipped with Algorithm 1. A nontrivial single-fault situation is when the faulty node i is not the source r and i has the value of its primary variable, i.e., its d-value, corrupted. In such a situation, there may be more than one node in the system (i.e., i and possibly some of its neighbors) that satisfy the condition $G(i)$. All of these nodes are privileged to make moves to change their d-values according to Algorithm 1. However, if a "wrong" node is selected first by the central demon to make a move to change its d-value, the fault will be spread in the system. The undesirable situation as in Table 1 may thus happen and cause the system to take a long time to self-stabilize. Therefore, more restrictions should somehow be imposed on Algorithm 1 to help (or force) the central demon to select the "right" node first to change its d-value. After the investigation of the non-trivial single-fault situation, we decide to classify the condition $G(i)$ into following cases.

Classification

1. $G(i) \wedge \exists j,k \in N(i)$ s.t. $j \neq k \wedge |d.i - d.j| > w(i,j) \wedge |d.i - d.k| > w(i,k)$ (This case will be referred to as Case 1 in the rest of the paper.)
2. $G(i) \wedge \exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)$ (The notation "$\exists!$" stands for "there exists a unique".)
   - 2.1. $j = r$
     - 2.1.1. $d.j = 0$ (Case 2)
     - 2.1.2. $d.j \neq 0$ (Case 3)
   - 2.2. $j \neq r$
     - 2.2.1. $d.i < d.j$
       - 2.2.1.2. $\exists u \in N(j) - \{i\}$ s.t. $|d.j - d.u| > w(j,u)$ (Case 4)
       - 2.2.1.1. $\forall u \in N(j) - \{i\}, |d.j - d.u| \leq w(j,u)$ (Case 5)
     - 2.2.2. $d.i > d.j$
       - 2.2.2.1. $G(j)$ (Case 6)
       - 2.2.2.2. $\neg G(j)$ (Case 7)
3. $G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k)$
   - 3.1. $c.i > 1$ (Case 8)
   - 3.2. $c.i = 1$
     - 3.2.1. $p.i = \perp$ (Case 9)
     - 3.2.2. $p.i = r$
       - 3.2.2.1. $d.r = 0$ (Case 10)
       - 3.2.2.2. $d.r \neq 0$ (Case 11)
     - 3.2.3. $p.i = j \neq r$
       - 3.2.3.1. $\neg G(j)$ (Case 12)
       - 3.2.3.2. $G(j)$
         - 3.2.3.2.2. $\exists u \in N(j) - \{i\}$ s.t. $|d.j - d.u| > w(j,u)$ (Case 13)
         - 3.2.3.2.1. $\forall u \in N(j) - \{i\}, |d.j - d.u| \leq w(j,u)$ (Case 14)

Corresponding to the above 14 cases, we obtain the following 12 lemmas (Lemmas 6∼17). In all these lemmas, it is assumed that the system starts in a legitimate state and incurs a single fault at a certain time instant t_0 and the fault corrupts the d-value of the faulty node and possibly its p-value and/or c-value too.

Lemma 6 (Corresponding to Case 1) *If at t_0^+, $\exists j,k \in N(i)$ s.t. $j \neq k \wedge |d.i - d.j| > w(i,j) \wedge |d.i - d.k| > w(i,k)$, then node i is the faulty node.*

Proof. Since $|d.i - d.j| > w(i,j)$ at t_0^+, either i or j is the faulty node by Lemma 3. Since $|d.i - d.k| > w(i,k)$ at t_0^+, either i or k is the faulty node by Lemma 3. Suppose i is not the faulty node. Then both j and k are faulty nodes. Since we are talking about the single-fault situation, we have a contradiction here. Therefore, i must be the faulty node. ■

Lemma 7 (Corresponding to Case 2) *If at t_0^+, $[\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)] \wedge j = r \wedge d.r = 0$, then i is the faulty node.*

Proof. Since $|d.i - d.r| > w(i,r)$ at t_0^+, either i or r is the faulty node by Lemma 3. Since $d.r = 0$ at t_0^+, r is not the faulty node. Therefore, i is the faulty node. ■

Lemma 8 (Corresponding to Case 3 and Case 11) *Suppose $i \neq r$. If $d.r \neq 0$ at t_0^+, then i is not the faulty node.*

Proof. *Since $d.r \neq 0$ at t_0^+, r is the faulty node and thus i is not the faulty node.* ■

Lemma 9 (Corresponding to Case 4 and Case 13) *Suppose $i \neq r$ and $j \in N(i)$. If $\exists u \in N(j) - \{i\}$ s.t. $|d.j - d.u| > w(j,u)$ at t_0^+, then i is not the faulty node.*

Proof. Since $|d.j - d.u| > w(u,j)$ at t_0^+, either j or u is the faulty node by Lemma 3. Thus, i is not the faulty node. ■

Lemma 10 (Corresponding to Case 5) *Suppose $i \neq r$. If at t_0^+, $G(i)$, $[\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)]$, $j \neq r$, $d.i < d.j$ and $[\forall u \in N(j) - \{i\}, |d.j - d.u| \leq w(j,u)]$, then node i is the faulty node.*

Proof. Since $|d.i - d.j| > w(i,j)$ and $[\forall u \in N(j) - \{i\}, |d.j - d.u| \leq w(j,u)]$ at t_0^+, we have $\exists! v \in N(j)$ s.t. $|d.j - d.v| > w(j,v)$ at t_0^+ and the unique v is just node i. Since $j \neq r$, $i \neq r$, $d.j > d.i$ and $G(i)$ at t_0^+, j is not the faulty node by the following Lemma 11. Since $|d.i - d.j| > w(i,j)$ at t_0^+, either i or j is the faulty node by Lemma 3. Therefore i is the faulty node. ■

Lemma 11 (Corresponding to Case 6) *Suppose $i \neq r$. If at t_0^+, $[\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)]$, $j \neq r$, $d.i > d.j$ and $G(j)$, then i is not the faulty node.*

Proof. *Suppose i is the faulty node. Since $|d.i - d.j| > w(i,j)$ and $d.i > d.j$ at t_0^+, we have $d.i > d.j + w(i,j)$ at t_0^+. On the other hand, $d.i = \min_{x \in N(i)}[d.x + w(i,x)] \leq d.j + w(i,j)$ at t_0^-. Hence, $d.i$ increases at t_0^+. If $d.i + w(j,i) \neq \min_{y \in N(j)}[d.y + w(j,y)]$ at t_0^-, then since $d.i$ increases at t_0^+, $\min_{y \in N(j)}[d.y + w(j,y)]$ does not change at t_0^+. Since $d.j = \min_{y \in N(j)}[d.y + w(j,y)]$ at t_0^- (for $j \neq r$) and $d.j$ does not change at t_0, we have $d.j = \min_{y \in N(j)}[d.y + w(j,y)]$ at t_0^+, i.e., $\neg G(j)$ at t_0^+, which causes a contradiction. Thus, $d.i + w(j,i) = \min_{y \in N(j)}[d.y + w(j,y)] = d.j$ at t_0^-. It follows that $d.j + w(i,j) = d.i + 2w(i,j) \neq d.i = \min_{x \in N(i)}[d.x + w(i,x)]$ at t_0^-. Let k be the node in $N(i)$, which satisfies $d.k + w(i,k) = \min_{x \in N(i)}[d.x + w(i,x)] = d.i$ at t_0^-. Then $k \neq j$. Since $d.i$ increases at t_0^+, $d.i - d.k > w(i,k)$ at t_0^+. Hence, $|d.i - d.k| > w(i,k)$ at t_0^+, which contradicts the condition $[\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)$ at $t_0^+]$. Therefore, i is not the faulty node.*

■

Lemma 12 (Corresponding to Case 7) *If at t_0^+, $[\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)]$, $j \neq r$ and $\neg G(j)$, then node i is the faulty node.*

Proof. Since, $|d.i - d.j| > w(i,j)$ at t_0^+, either i or j is the faulty node by Lemma 3. Since $\neg G(j)$ at t_0^+, j is not the faulty node by Lemma 4. Therefore, i is the faulty node. ■

Lemma 13 (Corresponding to Case 8) *Suppose $i \neq r$. If at t_0^+, $G(i)$, $[\forall k \in N(i), |d.i - d.k| \leq w(i,k)]$ and $c.i > 1$, then node i is the faulty node.*

Proof. Suppose i is not the faulty node. Since $G(i)$ at t_0^+, the faulty node must be some node u in $N(i)$ by Lemma 5. Since $d.i = \min_{x \in N(i)}[d.x + w(i,x)]$ at t_0^-, $d.i \neq \min_{x \in N(i)}[d.x + w(i,x)]$ at t_0^+ and $d.i$ does not change at t_0^+ (for i is not the faulty node), we have $\min_{x \in N(i)}[d.x + w(i,x)]$ changes at t_0^+. Since $c.i > 1$ at t_0^+ and $c.i$ does not change at t_0 (for i is not the faulty node), we have $c.i > 1$ at t_0^-. Hence, there exist two distinct neighbors j, k of i such that $d.j + w(i,j) = \min_{x \in N(i)}[d.x + w(i,x)] = d.k + w(i,k)$ at t_0^-. If $d.u$ increases at t_0^+, then $\min_{x \in N(i)}[d.x + w(i,x)]$ does not change at t_0^+, which causes a contradiction. Hence, $d.u$ must decrease at t_0^+ so that $d.u + w(i,u) = \min_{x \in N(i)}[d.x + w(i,x)]$ at t_0^+, and thus $\min_{x \in N(i)}[d.x + w(i,x)]$ also decreases at t_0^+. Since $d.i = \min_{x \in N(i)}[d.x + w(i,x)]$ at t_0^-, $d.i > \min_{x \in N(i)}[d.x + w(i,x)] = d.u + w(i,u)$ at t_0^+. Hence, $|d.i - d.u| > w(i,u)$ at t_0^+, which causes a contradiction. Therefore, i is the faulty node. ■

Lemma 14 (Corresponding to Case 9) *Suppose $i \neq r$. If at t_0^+, $c.i = 1$ and $p.i = \perp$, then node i is the faulty node.*

Proof. Suppose i is not the faulty node. Since $c.i = 1$ at t_0^+, we have $c.i = 1$ at t_0^-. Hence, $p.i \neq \perp$ at t_0^- and hence at t_0^+, which causes a contradiction. Therefore, i is the faulty node. ■

Lemma 15 (Corresponding to Case 10) *Suppose $i \neq r$. If at t_0^+, $G(i)$, $[\forall k \in N(i), |d.i - d.k| \leq w(i,k)]$, $c.i = 1$, $p.i = r$ and $d.r = 0$, then node i is the faulty node.*

Proof. Suppose i is not the faulty node. Since $p.i = r$ at t_0^+, $p.i = r$ at t_0^-. Hence, $d.r + w(i,r) = \min_{x \in N(i)} [d.x + w(i,x)] = d.i$ at t_0^-. Since $d.r = 0$ at t_0^+, r is not the faulty node. Thus, $d.r + w(i,r) = d.i$ at t_0^+. Consequently, $\min_{x \in N(i)} [d.x + w(i,x)] \leq d.i$ at t_0^+. Since $G(i)$, i.e., $\min_{x \in N(i)} [d.x + w(x,i)] \neq d.i$, at t_0^+, $\min_{x \in N(i)} [d.x + w(i,x)] < d.i$ at t_0^+. Hence, there is a $j \neq r$ such that $d.j + w(i,j) < d.i$ at t_0^+ and thus $|d.i - d.j| = d.i - d.j > w(i,j)$ at t_0^+, which causes a contradiction. Therefore, node i is the faulty node. ■

Lemma 16 (Corresponding to Case 12) *Suppose $i \neq r$. If at t_0^+, $G(i)$, $[\forall k \in N(i), |d.i - d.k| \leq w(i,k)]$, $c.i = 1$, $p.i = j \neq r$ and $\neg G(j)$, then node i is the faulty node.*

Proof. By Lemma 5, the faulty node must be node i or one of i's neighbors. Since $\neg G(j)$ at t_0^+, j is not the faulty node by Lemma 4. Hence, the faulty node is in $N(i) \cup \{i\} - \{j\}$. Suppose i is not the faulty node. Since $p.i = j$ at t_0^+, $p.i = j$ at t_0^-. Thus, $d.j + w(i,j) = \min_{x \in N(i)} [d.x + w(i,x)] = d.i$ at t_0^-. Since neither i nor j is the faulty node, $d.i = d.j + w(i,j)$ at t_0^+ still. Since $G(i)$, i.e., $d.i \neq \min_{x \in N(i)} [d.x + w(i,x)]$, at t_0^+ and $\min_{x \in N(i)} [d.x + w(i,x)] \leq d.j + w(i,j) = d.i$ at t_0^+, we have $\min_{x \in N(i)} [d.x + w(i,x)] < d.i$ at t_0^+. Hence, $\exists l \in N(i) - \{j\}$ s.t. $d.l + w(i,l) < d.i$ at t_0^+. It follows that $|d.i - d.l| = d.i - d.l > w(i.l)$ at t_0^+, which causes a contradiction. Therefore, i is the faulty node. ■

Lemma 17 (Corresponding to Case 14) *It can not happen that $\exists i \neq r$ such that $G(i)$, $[\forall k \in N(i), |d.i - d.k| \leq w(i,k)]$, $c.i = 1$, $p.i = j \neq r$, $G(j)$ and $\forall u \in N(j) - \{i\}, |d.j - d.u| \leq w(j,u)$ at t_0^+. In other words, Case 14 cannot happen in a single-fault situation.*

Proof. Suppose $\exists i \neq r$ satisfying above conditions. Then since $G(i)$ and $G(j)$ at t_0^+, the faulty node must be in both $N(i) \cup \{i\}$ and $N(j) \cup \{j\}$ in view of Lemma 5. Hence, the faulty node is in $(N(i) \cup \{i\}) \cap (N(j) \cup \{j\}) = (N(i) \cap N(j)) \cup \{i,j\}$. Thus, we have three cases to consider.

Case 1. The faulty node is in $N(i) \cap N(j)$.

Let l be the faulty node. Thus $l \neq i$, $l \neq j$ and thus i is not the faulty node. Since $c.i = 1$ and $p.i = j$ at t_0^+, we have $c.i = 1$ and $p.i = j$ at t_0^-. Hence, $|S(i)| = 1$ and $j \in S(i)$ at t_0^-. It follows that $l \notin S(i)$ at t_0^-, that is, $d.l + w(i,l) \neq \min_{x \in N(i)} [d.x + w(i,x)]$ at t_0^-. Since $d.i$ does not change at t_0^+ and $d.i = \min_{x \in N(i)} [d.x + w(i,x)]$ at t_0^- and $d.i \neq \min_{x \in N(i)} [d.x + w(i,x)]$ at t_0^+, $\min_{x \in N(i)} [d.x + w(i,x)]$ must changes at t_0^+. Thus, $d.l$ decreases at t_0^+ so that $\min_{x \in N(i)} [d.x + w(i,x)] = d.l + w(i,l)$ decreases at t_0^+. Since $d.i$ does not change at t_0^+, $\min_{x \in N(i)} [d.x + w(i,x)] < d.i$ at t_0^+. Hence, $d.l + w(i,l) < d.i$ at t_0^+ and hence $|d.i - d.l| = d.i - d.l > w(i,l)$ at t_0^+, which causes a contradiction.

Case 2. Node i is the faulty node.

Since $p.i = j$ at t_0^-, we have $d.j + w(i,j) = \min_{x \in N(i)} [d.x + w(i,x)] = d.i$ at t_0^-. If $d.i$ increases at t_0^+, then $d.j + w(i,j) < d.i$ at t_0^+. Thus, $|d.i - d.j| = d.i - d.j > w(i,j)$ at t_0^+, which causes a contradiction. It follows that $d.i$ decreases at t_0^+. Since $d.j = \min_{y \in N(j)} [d.y + w(j,y)]$ at t_0^- and $d.j \neq \min_{y \in N(j)} [d.y + w(j,y)]$ at t_0^+, $\min_{y \in N(j)} [d.y + w(j,y)]$ must change at t_0^+. Hence, $\min_{y \in N(j)} [d.y + w(j,y)]$ decreases at t_0^+ and $d.i + w(i,j) = \min_{y \in N(j)} [d.y + w(j,y)] < d.j$ at t_0^+. Consequently, $|d.i - d.j| = d.j - d.i > w(i,j)$ at t_0^+, which causes a contradiction.

Case 3. Node j is the faulty node.

By the same argument as in Case 2 above, we are led to a contradiction.

Therefore, there does not exist an $i \neq r$ satisfying all those conditions in the statement of Lemma 17. ■

In the above, all the non-trivial single-fault situations have been analyzed. The following lemma makes the situation even more transparent.

Lemma 18 *If i is not the faulty node and executes $R0$ or $R1$ at t_1, where t_1 is the first time instant after t_0 at which the system executes $R0$ or $R1$ to change the d-value, then the d-values of the whole system can not get back to normal at t_1^+.*

Proof. Let u be the faulty node. Then the value of $d.u$ at t_1^+ is not equal to the value of $d.u$ at t_0^-. Hence, the global state of the system at t_1^+ is not equal to the global state of the system at t_0^-. Since there is a unique legitimate state in the system, the global state of the system at t_1^+ is not the legitimate state. ■

With the help of the above analysis, it is now clear how restrictions should be imposed on Algorithm 1 in order for the system to contain the fault after it incurs a single-fault situation. Explicitly, any node that satisfies the condition in Case 3, Case 4, Case 6, Case 11 or Case 13 should be prohibited from making a move to change its d-value. Thus, we obtain the following prototype for our fault-containing algorithm.

Prototype

{For the source r}

$R0 : d.r \neq 0 \rightarrow d.r := 0$

{For node $i \neq r$}

$R1 : G(i) \wedge \exists j, k \in N(i)$ s.t. $j \neq k \wedge |d.i - d.j| > w(i,j) \wedge |d.i - d.k| > w(i,k) \rightarrow A(i)$ (This corresponds to Case 1 in the above classification.)

$R2 : G(i) \wedge [\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)] \wedge j = r \wedge d.j = 0 \rightarrow A(i)$ (This corresponds to Case 2.)

$R3 : G(i) \wedge [\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)] \wedge j \neq r \wedge d.i < d.j \wedge \forall u \in N(j) - \{i\}, |d.j - d.u| \leq w(u,j) \rightarrow A(i)$ (This corresponds to Case 5.)

$R4 : G(i) \wedge [\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)] \wedge j \neq r \wedge d.i > d.j \wedge \neg G(j) \rightarrow A(i)$ (This corresponds to Case 7.)

$R5 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge c.i > 1 \rightarrow A(i)$ (This corresponds to Case 8.)

$R6 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge c.i = 1 \wedge p.i = \perp \rightarrow A(i)$ (This corresponds to Case 9.)

$R7 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge c.i = 1 \wedge p.i = r \wedge d.r = 0 \rightarrow A(i)$ (This corresponds to Case 10.)

$R8 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge c.i = 1 \wedge p.i = j \neq r \wedge \neg G(j) \rightarrow A(i)$ (This corresponds to Case 12.)

$R9 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge c.i = 1 \wedge p.i = j \neq r \wedge G(j) \wedge \forall u \in N(j) - \{i\}, |d.j - d.u| \leq w(j,u) \rightarrow A(i)$ (This corresponds to Case 14.)

$R10 : \neg G(i) \wedge c.i \neq |S(i)| \rightarrow c.i := |S(i)|$ (This is $R2$ in Algorithm 1.)

$R11 : \neg G(i) \wedge c.i = |S(i)| = 1 \wedge p.i \neq j$, where j is the unique node in $S(i) \rightarrow p.i := j$ (This is $R3$ in Algorithm 1.)

$R12 : \neg G(i) \wedge c.i = |S(i)| > 1 \wedge p.i \neq \perp \rightarrow p.i := \perp$ (This is $R4$ in Algorithm 1.)

The legitimate state is the same as that for Algorithm 1.

The above analysis (i.e., Lemmas 6∼18) has already convinced us that the Prototype has the fault-containment property. Before transforming Prototype into a distributed algorithm, we should also check the no-deadlock property for it, because the imposition of restrictions to a self-stabilizing algorithm may cause a deadlock to the system.

Lemma 19 (No deadlock) *At the prototype level, the system is never deadlocked in an illegitimate state.*

Proof. Suppose the system is in an illegitimate state. Then $d.r \neq 0 \vee [\exists i \neq r$ s.t. $G(i) \vee c.i \neq |S(i)| \vee (c.i = |S(i)| = 1 \wedge p.i \neq j$, where j is the unique node in $S(i)) \vee (c.i = |S(i)| > 1 \wedge p.i \neq \perp)]$.

Case 1. $d.r \neq 0$. Then r can execute $R0$.

Case 2. $d.r = 0$ and $\forall y \neq r$, $\neg G(y)$. Then $\exists i \neq r$ s.t. $c.i \neq |S(i)|$, $(c.i = |S(i)| = 1 \wedge p.i \neq j)$ or $(c.i = |S(i)| > 1 \wedge p.i \neq \perp)$.

Subcase 2.1. $\exists i \neq r$ s.t. $c.i \neq |S(i)|$. Then i can execute $R10$.

Subcase 2.2. $\exists i \neq r$ s.t. $c.i = |S(i)| = 1 \wedge p.i \neq j$, where j is the unique node in $S(i)$. Then i can execute $R11$.

Subcase 2.3. $\exists i \neq r$ s.t. $c.i = |S(i)| > 1 \wedge p.i \neq \perp$. Then i can execute $R12$.

Case 3. $d.r = 0$ and $\exists i \neq r$ s.t. $G(i)$.

Subcase 3.1. $\exists i \neq r$ s.t. $G(i)$ and $\exists j, k \in N(i)$ s.t. $j \neq k \wedge |d.i - d.j| > w(i,j) \wedge |d.i - d.k| > w(i,k)$. Then i can execute $R1$.

Subcase 3.2. $\forall y \neq r, [G(y) \rightarrow$ there is at most one node $z \in N(y)$ s.t. $|d.z - d.y| > w(z,y)]$.

Subcase 3.2.1. $\exists i \neq r$ s.t. $G(i)$ and $\exists! j \in N(i)$ s.t. $|d.j - d.i| > w(j,i)$. Thus, $d.i \neq d.j$.

Subcase 3.2.1.2. $d.i < d.j$. Then $d.j > 0$ and hence $j \neq r$ (in view of the condition for Case 3). Since $d.j > d.i$ and $|d.j - d.i| > w(j,i)$, we have $d.i + w(j,i) < d.j$. Hence, $\min_{x \in N(j)} [d.x + w(j,x)] \neq d.j$, that is, $G(j)$. Since $|d.j - d.i| > w(j,i)$, we have $\forall u \in N(j) - \{i\}, |d.u - d.j| \leq w(u,j)$ in view of the condition for Subcase 3.2. Thus, i can execute $R3$.

Subcase 3.2.1.1. $d.i > d.j$.

Subcase 3.2.1.1.1. $j = r$. Then $d.j = 0$ in view of the condition for Case 3. Hence, i can execute $R2$.

Subcase 3.2.1.1.2. $j \neq r$ and $G(j)$. Since $|d.j - d.i| > w(j,i)$, we have $\forall k \in N(j) - \{i\}, |d.k - d.j| \leq w(k,j)$ in view of the condition for Subcase 3.2. Hence, j can execute $R3$.

Subcase 3.2.1.1.3. $j \neq r$ and $\neg G(j)$. Then i can execute $R4$.

Subcase 3.2.2. $\forall y \neq r, [G(y) \rightarrow \forall z \in N(y), |d.z - d.y| \leq w(z,y)]$. Then let $i \neq r$ such that $G(i)$. Thus, $\forall j \in N(i), |d.j - d.i| \leq w(j,i)$.

Subcase 3.2.2.1. $c.i > 1$. Then i can execute $R5$.

Subcase 3.2.2.2. $c.i = 1 \wedge p.i = \perp$. Then i can execute $R6$.

Subcase 3.2.2.3. $c.i = 1 \wedge p.i = r$. Since $d.r = 0$, i can execute $R7$.

Subcase 3.2.2.4. $c.i = 1 \wedge p.i = j \neq r$.

Subcase 3.2.2.4.1. $\neg G(j)$. Then i can execute $R8$.

Subcase 3.2.2.4.2. $G(j)$. In view of the condition for Subcase 3.2.2, we see that $\forall u \in N(j), |d.u - d.j| \leq w(u,j)$. Hence, i can execute $R9$.

Thus, we have considered all cases of illegitimate states, and we have shown that in any case, there always exists a node in the system which is privileged to make a move. Therefore, the lemma is proven. ■

Note that some rules in the above Prototype require a node to collect information from neighbors of distance 2 (e.g., in the guard condition of $R4$, node j is a neighbor of node i, and i needs to know information about j's neighbors). However, in a distributed system, a node is not allowed to read information of nodes other than its direct neighbors. Therefore, in order for us to transform

Prototype into a distributed algorithm, auxiliary secondary variables q_1, q_2 (question) and a_1, a_2 (answer) need to be used to fulfill the job of collecting information. (The idea of applying auxiliary variables q_1, q_2, a_1 and a_2 is attributed to Ghosh *et al.* [6, 8, 9, 10] and Gupta [11].) Thus, our fault-containing algorithm is finally ready.

Algorithm 2

```
{For the source $r$}
  $R0 : d.r \neq 0 \to d.r := 0$ (This is $R0$ in Prototype.)

{For node $i \neq r$}
  $R1 : G(i) \wedge \exists j, k \in N(i)$ s.t. $j \neq k \wedge |d.i - d.j| > w(i,j)$
      $\wedge |d.i - d.k| > w(i,k) \to A(i)$ (This is $R1$ in
      Prototype.)
  $R2 : G(i) \wedge [\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)] \wedge j = r$
      $\wedge d.j = 0 \to A(i)$ (This is $R2$ in Prototype.)
  $R3 : G(i) \wedge [\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)] \wedge j \neq r$
      $\wedge a_1.j = 0 \wedge q_1.i = 0 \to q_1.i := 1$ (Rules $R3$, $R4$,
      $R5$ and $R6$ here are devised to implement $R3$ and
      $R4$ in Prototype.)
  $R4 : [\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)] \wedge j \neq r \wedge$
      $d.j < d.i \wedge q_1.j = 1 \wedge a_1.i = 0 \to a_1.i := 1$
  $R5 : \neg G(i) \wedge \exists j \in N(i) - \{r\}$ s.t. $[|d.i - d.j| > w(i,j) \wedge$
      $d.j > d.i \wedge q_1.j = 1] \wedge a_1.i = 0 \to a_1.i := 1$
  $R6 : G(i) \wedge [\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)] \wedge j \neq r$
      $\wedge q_1.i = 1 \wedge a_1.j = 1 \to A(i)$
  $R7 : \neg G(i) \wedge q_1.i = 1 \to q_1.i := 0$
  $R8 : \forall j \in N(i) - \{r\}, [|d.i - d.j| \leq w(i,j) \vee q_1.j = 0]$
      $\wedge a_1.i = 1 \to a_1.i := 0$
  $R9 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge c.i > 1 \to$
      $A(i)$ (This is $R5$ in Prototype.)
  $R10 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge c.i = 1 \wedge$
      $p.i = \perp \to A(i)$. (This is $R6$ in Prototype.)
  $R11 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge c.i = 1$
      $\wedge p.i = r \wedge d.r = 0 \to A(i)$ (This is $R7$ in
      Prototype.)
  $R12 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge c.i = 1 \wedge$
      $p.i = j \neq r \wedge a_2.j = 0 \wedge q_2.i = 0 \to q_2.i := 1$
      (Rules $R13$, $R14$, $R15$ and $R16$ here are devised
      to implement $R8$ and $R9$ in Prototype.)
  $R13 : \neg G(i) \wedge \exists j \in N(i) - \{r\}$ s.t. $[p.j = i \wedge q_2.j = 1$
      $\wedge c.j = 1 \wedge |d.i - d.j| \leq w(i,j)] \wedge a_2.i = 0 \to$
      $a_2.i := 1$
  $R14 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge \exists j \in$
      $N(i) - \{r\}$ s.t. $[p.j = i \wedge q_2.j = 1 \wedge c.j = 1] \wedge$
      $a_2.i = 0 \to a_2.i := 1$
  $R15 : G(i) \wedge \forall k \in N(i), |d.i - d.k| \leq w(i,k) \wedge c.i = 1 \wedge$
      $p.i = j \neq r \wedge q_2.i = 1 \wedge a_2.j = 1 \to A(i)$
  $R16 : \neg G(i) \wedge q_2.i = 1 \to q_2.i := 0$
  $R17 : \forall j \in N(i) - \{r\}, q_2.j = 0 \wedge a_2.i = 1 \to a_2.i := 0$
  $R18 : \neg G(i) \wedge c.i \neq |S(i)| \to c.i := |S(i)|$ (This is $R10$
      in Prototype.)
  $R19 : \neg G(i) \wedge c.i = |S(i)| = 1 \wedge p.i \neq j \to p.i := j$,
      where $j$ is the unique node in $S(i)$ (This is $R1$
      in Prototype.)
  $R20 : \neg G(i) \wedge c.i = |S(i)| > 1 \wedge p.i \neq \perp \to p.i := \perp$
      (This is $R12$ in Prototype.)
```

Note that in the above algorithm, all of the four variables q_1, q_2, a_1 and a_2 take values in the set $\{0, 1\}$. Legitimate states are defined to be all those global states in which the following condition holds:

$d.r = 0 \wedge \forall i \neq r, \neg G(i) \wedge q_1.i = q_2.i = a_1.i = a_2.i = 0 \wedge c.i = |S(i)| \wedge [(|S(i)| = 1 \wedge p.i = j$, where j is the unique node in $S(i)) \vee (|S(i)| > 1 \wedge p.i = \perp)]$.

To help readers comprehend the relationship between Algorithm 2 and the Prototype, we explain it in more detail here. For instance, rules $R3$, $R4$, and $R6$ in Algorithm 2 are devised to implement rule $R3$ in Prototype. When a node i satisfies the condition $[G(i) \wedge [\exists! j \in N(i)$ s.t. $|d.i - d.j| > w(i,j)] \wedge j \neq r \wedge d.i < d.j]$, it wants to know if node j satisfies the condition $[\forall u \in N(j) - \{i\}, |d.u - d.j| \leq w(u,j)]$. According to $R3$ in Algorithm 2, it sets $q_1.i = 1$ to ask j whether j satisfies the condition. If j satisfies the condition, then, according to rule $R4$ in Algorithm 2, it sets $a_1.j = 1$ to let i know that its answer is affirmative. Then according to rule $R6$ in Algorithm 2, node i can make a move to change its d-value. Thus, rule $R3$ in the Prototype is faithfully implemented. Note that in rule $R3$ of Algorithm 2, the condition $a_1.j = 0$ is placed there as a requisite. This is a little subtle. One can see that if the condition is taken away from $R3$, then node i may change its q_1-value to 1 in a situation where j does not satisfy the condition and yet $a_1.j = 1$ due to whatever reason. Node i may then go ahead to execute $R6$ and this is not the intended result from $R3$ in Prototype.

In the end of this section, we outline the main results concerning the correctness and the efficiency of our proposed algorithm. They are covered by the following theorems. The proofs of these theorems are not provided here. They are quite long and we intend to have them presented elsewhere.

Theorem 20 (No deadlock) *The system is never deadlocked in an illegitimate state.*

Theorem 21 (Self-stabilization) *Starting with any initial state, the system will eventually stop in the legitimate state.*

Theorem 22 (Fault containment) *From any single-fault state till reaching a legitimate state, the system changes the d-value at most once.*

Theorem 23 (Stabilization time) *For single-fault situations, the stabilization time of the algorithm is $O(\Delta)$, where Δ is the maximum node degree.*

4 An illustration

The example in Figure 3 is to illustrate the execution of Algorithm 2. Note that in each configuration, the shaded nodes represent privileged nodes, whereas the shaded node with a darkened circle stands for the privileged node selected by the central demon to make a move.

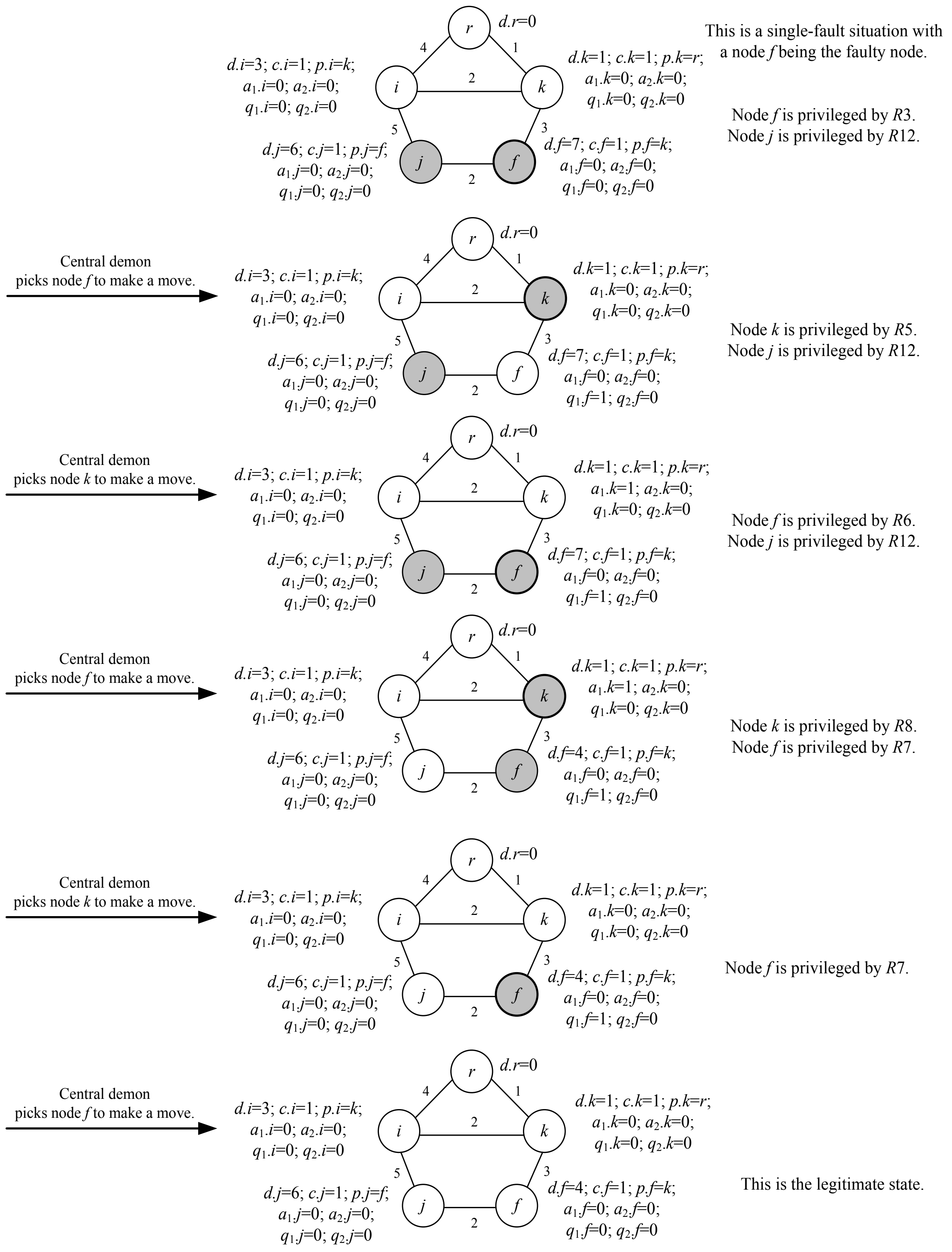
d.r=0
d.i=3; c.i=1; p.i=k; a1.i=0; a2.i=0; q1.i=0; q2.i=0
d.k=1; c.k=1; p.k=r; a1.k=0; a2.k=0; q1.k=0; q2.k=0
d.j=6; c.j=1; p.j=f; a1.j=0; a2.j=0; q1.j=0; q2.j=0
d.f=7; c.f=1; p.f=k; a1.f=0; a2.f=0; q1.f=0; q2.f=0
This is a single-fault situation with a node f being the faulty node.
Node f is privileged by R3.
Node j is privileged by R12.
Central demon picks node f to make a move.
d.f=7; c.f=1; p.f=k; a1.f=0; a2.f=0; q1.f=1; q2.f=0
Node k is privileged by R5.
Node j is privileged by R12.
Central demon picks node k to make a move.
d.k=1; c.k=1; p.k=r; a1.k=1; a2.k=0; q1.k=0; q2.k=0
Node f is privileged by R6.
Node j is privileged by R12.
Central demon picks node f to make a move.
d.f=4; c.f=1; p.f=k; a1.f=0; a2.f=0; q1.f=1; q2.f=0
Node k is privileged by R8.
Node f is privileged by R7.
Central demon picks node k to make a move.
Node f is privileged by R7.
Central demon picks node f to make a move.
d.f=4; c.f=1; p.f=k; a1.f=0; a2.f=0; q1.f=0; q2.f=0
This is the legitimate state.

Figure 3: An example which illustrates the execution of Algorithm 2.

5 Concluding remarks

In the above, we have proposed a fault-containing self-stabilizing algorithm for the shortest path problem. The above work has two implications: Firstly, the approach in [17] is tested again as valid. Secondly, the improvement made by the modified algorithm upon the original algorithm in [1] and [15] can demonstrate the desirability of a well-designed fault-containing self-stabilizing algorithm. We would like to point out that our algorithm is more general than the fault-containing BFS-tree-finding algorithm in [10]. Moreover, for the algorithm in [10], we have computed the worst-case stabilization time for single-fault situations to be $\Theta(\Delta^2)$, where Δ is the maximum node degree in the system; and in this respect, our algorithm is faster, with $O(\Delta)$ as its worst-case stabilization time for single-fault situations. Our computation of the worst-case stabilization time for single-fault situations for the algorithm in [10] is conducted as follows.

Let $G = (V, E)$ be a distributed system with n processors. Let r be a distinguished processor in G. $\forall i \in V$, i maintains a primary variable l_i and $\forall i \neq r$, i also maintains another primary variable p_i. $\forall i \in V$ and $\forall j \in N(i)$, i maintains auxiliary variables q_{ij} and a_{ij}. The variable l_i takes values in $\{1, 2, \cdots, n\}$, the variable p_i takes values in $N(i)$, the variable q_{ij} takes values in $\{ask, \perp\}$ and values taken by the variable a_{ij} will be understood in the following algorithm. The following is the main part (i.e., the rules for processor $i \neq r$) of the fault-containing BFS-tree algorithm in Ghosh *et al.* [10]. We omit the part for processor r because it has nothing to do with the computation to be carried out.

```
{For processor $i \neq r$}
  [$S_1$] $(l_i \neq l_{p_i} + 1) \wedge (l_{p_i} < n) \rightarrow l_i := l_{p_i} + 1$
  [$S_2$] $\exists k \in N(i) : l_k = \min\{ l_i \mid j \in N(i)\} \wedge l_{p_i} > l_k \rightarrow$
          $l_i := l_{k_i} + 1;\ p_i = k;$
  [$S_5$] $\exists j \in N(i) : trigger^1_{ij} \wedge q^1_{ij} = \perp \wedge a^1_{ji} = \perp \rightarrow q^1_{ij} :=$
          $ask$
  [$S_6$] $\exists j \in N(i) : a^1_{ij} \neq f^1_i\ (q^1_{ji}\ , neighborhood(i)) \rightarrow$
          $a^1_{ij} := f^1_i\ (q^1_{ji}\ , neighborhood(i))$
  [$S_7$] $\exists j \in N(i) : \neg trigger^1_{ij} \wedge q^1_{ij} \neq \perp \rightarrow q^1_{ij} := \perp$
  [$S_5{}'$] $\exists j \in N(i) : trigger^2_{ij} \wedge q^2_{ij} = \perp \wedge a^2_{ji} = \perp \rightarrow q^2_{ij} :=$
          $ask$
  [$S_6{}'$] $\exists j \in N(i) : a^2_{ij} \neq f^2_i\ (q^2_{ji}\ , neighborhood(i)) \rightarrow$
          $a^2_{ij} := f^2_i\ (q^2_{ji}\ , neighborhood(i))$
  [$S_7{}'$] $\exists j \in N(i) : \neg trigger^2_{ij} \wedge q^2_{ij} \neq \perp \rightarrow q^2_{ij} := \perp$
  [$S_8$] $G_p(i) \wedge \neg Q_1(i) \wedge can\_stabilize(i) \rightarrow stabilize(i)$
  [$S_9$] $G_p(i) \wedge [\neg Q_1(i) \wedge \neg can\_stabilize(i)] \wedge [\neg Q_2(i) \wedge$
          $\forall j \in N(i) : \neg can\_stabilize(j)]$
           $\rightarrow$ if
                $G_1(i)\ \rightarrow A_1(i)$
                $G_2(i)\ \rightarrow A_2(i)$
              fi
where
  $G_1(i)$ : the guard in $S_1$
  $G_2(i)$ : the guard in $S_2$
  $A_1(i)$ : the action part in $S_1$
  $A_2(i)$ : the action part in $S_2$
  $G_p(i) \equiv G_1(i) \vee G_2(i)$
  $can\_stabilize(i) \equiv G_p(i) \wedge$ execution of $stabilize(i) \Rightarrow$
  $(\neg G_p(i) \wedge \forall j \in N(i), \neg G_p(j))$
  $stabilize(i) \equiv$ if $G_2(i)$ then $A_2(i)$ else $A_1(i)$
  $trigger^1_{ij} \equiv G_p(i)$
  $trigger^2_{ij} \equiv G_p(i) \wedge (\forall k \in N(i), q^1_{ik} = ask \wedge a^1_{ij} \neq \perp) \wedge$
  $\neg can\_stabilize(i)$
  $f^1_i = \begin{cases} \langle l_{p_i}, x\rangle, & \text{if } q^1_{ji} = ask; \\ \perp, & otherwise; \end{cases}$
  where $x = \min\{\ l_k \mid k \in N(i) - \{j\}\ \}$
  $phase\_1\_over(i) \equiv \neg G_p(i) \vee (\forall k \in N(i), q^1_{ik} = ask \wedge$
  $a^1_{ki} \neq \perp)$
  $f^2_i = \begin{cases} 0, & \text{if } q^2_{ji} = ask \wedge phase\_1\_over(i) \wedge \\ & \quad \neg can\_stabilize(i); \\ 1, & \text{if } q^2_{ji} = ask \wedge phase\_1\_over(i) \wedge \\ & \quad can\_stabilize(i); \\ \perp, & otherwise; \end{cases}$
  $Q_1(i) : \exists j \in N(i) : trigger^1_{ij} \wedge (q^1_{ij} = ask \wedge a^1_{ji} = \perp)$
  $Q_2(i) : \exists j \in N(i) : trigger^2_{ij} \wedge (q^2_{ij} = ask \wedge a^2_{ji} = \perp)$
```

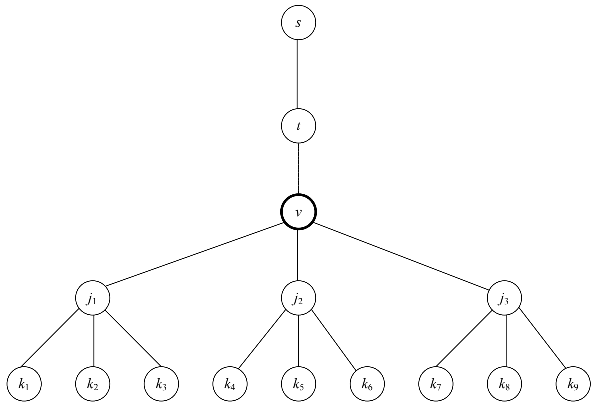

Figure 4: A system with the maximum node degree 4

A legitimate state is a state in which $G_p(i)$ is false for every node i, and $q^1_{ij} = q^2_{ij} = a^1_{ij} = a^2_{ij} = \perp$ for every node i and every node j.

We consider the system in Figure 4 in which the maximum node degree is 4. Let us start with the single-fault state [$l_s = 1$, $l_t = 2$, $l_v = 4$, $l_{j_1} = l_{j_2} = l_{j_3} = 4$, $l_{k_1} = l_{k_2} = l_{k_3} = l_{k_4} = l_{k_5} = l_{k_6} = l_{k_7} = l_{k_8} = l_{k_9} = 5$, and $q^1_{ij} = q^2_{ij} = a^1_{ij} = a^2_{ij} = \perp$ $\forall i$ and $\forall j \in N(i)$] in which node v is the faulty node. Then the system can move in the following sequence:

1.1) The central demon selects node v to execute S_5 and change the value of $q^1_{vj_1}$ from $\perp$ to ask.

1.2) The central demon selects node v to execute S_5 and change the value of $q^1_{vj_2}$ from $\perp$ to ask.

1.3) The central demon selects node v to execute S_5 and change the value of $q^1_{vj_3}$ from $\perp$ to ask.

1.4) The central demon selects node v to execute S_5 and change the value of q^1_{vp} from $\perp$ to ask.

2.1) The central demon selects node j_1 to execute S_5 and change the value of $q^1_{j_1k_1}$ from $\perp$ to ask.

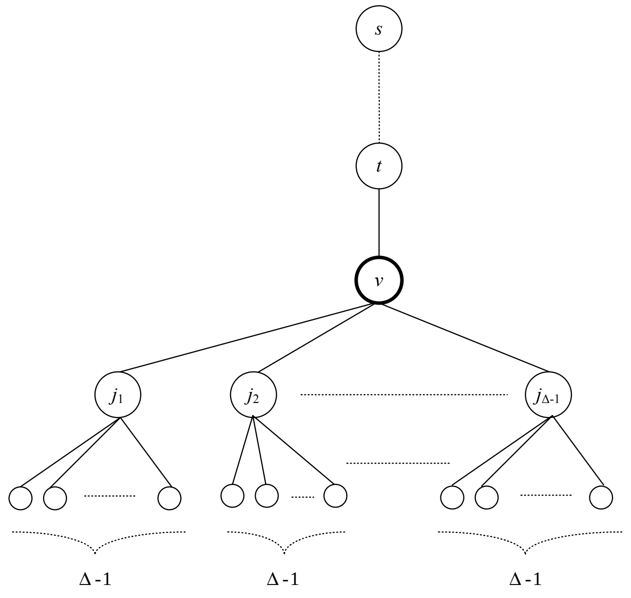

Figure 5: A system with the maximum node degree Δ

2.2) The central demon selects node j_1 to execute S_5 and change the value of $q^1_{j_1k_2}$ from $\perp$ to *ask*.
2.3) The central demon selects node j_1 to execute S_5 and change the value of $q^1_{j_1k_3}$ from $\perp$ to *ask*.
2.4) The central demon selects node j_1 to execute S_5 and change the value of $q^1_{j_1i}$ from $\perp$ to *ask*.
3.1) The central demon selects node j_2 to execute S_5 and change the value of $q^1_{j_2k_4}$ from $\perp$ to *ask*.
3.2) The central demon selects node j_2 to execute S_5 and change the value of $q^1_{j_2k_5}$ from $\perp$ to *ask*.
3.3) The central demon selects node j_2 to execute S_5 and change the value of $q^1_{j_2k_6}$ from $\perp$ to *ask*.
3.4) The central demon selects node j_2 to execute S_5 and change the value of $q^1_{j_2i}$ from $\perp$ to *ask*.
4.1) The central demon selects node j_3 to execute S_5 and change the value of $q^1_{j_3k_7}$ from $\perp$ to *ask*.
4.2) The central demon selects node j_3 to execute S_5 and change the value of $q^1_{j_3k_8}$ from $\perp$ to *ask*.
4.3) The central demon selects node j_3 to execute S_5 and change the value of $q^1_{j_3k_9}$ from $\perp$ to *ask*.
4.4) The central demon selects node j_3 to execute S_5 and change the value of $q^1_{j_3i}$ from $\perp$ to *ask*.

Note that 16 moves have been made and the system has not yet reached the legitimate state. We now generalize this example to the system in Figure 5. The maximum node degree of the system is Δ. Let us start with a single-fault state in which $l_s = 1, \cdots, l_t = k-1$, $l_v = k+1$, $l_{j_1} = l_{j_2} = \cdots = l_{j_{\Delta-1}} = k+1$ and the l-values of all the bottom nodes are $k+2$ (thus, v is the faulty node). If we let the system move in the similar fashion as in the preceding example, it is not difficult to see that the system has not yet reached the legitimate state after it has made Δ^2 moves. From all the above, one can see that the worst-case stabilization time of the protocol in [10] for single-fault situations is $\Omega(\Delta^2)$. The verification, that the worst-case stabilization time of the above protocol for single-fault situations is $O(\Delta^2)$, is omitted.

References

[1] S. Chandrasekar and P.K. Srimani, "A self-stabilizing algorithm for all-pairs shortest path tree problem", Parallel Algorithm and Applications, Vol. 4(1&2), pp.125-137, 1994.

[2] E.W. Dijkstra, "Self-stabilizing systems in spite of distributed control", Comm. ACM 17(11), pp.643-644, 1974.

[3] E.W. Dijkstra, "Self-stabilizing systems in spite of distributed control (EWD391)", Reprinted in: Selected writing on computing: a personal perspective. Berlin Heidelberg New York Springer, pp.41-46, 1982.

[4] E.W. Dijkstra, "A belated proof of self-stabilization", Distributed Computing, Vol. 1(1), pp.5-6, 1986.

[5] S. Dolev, A. Israeli and S. Moran, "Self-stabilization of dynamic systems assuming only read/write atomicity", Distributed Computing, Vol. 7(1), pp.3-16, 1993.

[6] S. Ghosh and A. Gupta, "An exercise in fault-containment: self-stabilizing leader election", Information Processing Letters, Vol. 59, pp.281-288, 1996.

[7] S. Ghosh, A. Gupta and T. Herman, "Fault-containing self-stabilizing distributed protocols", Technical Report 00-01, Department of Computer Science, The University of Iowa, Iowa City, 2000.

[8] S. Ghosh, A. Gupta, T. Herman and S.V. Pemmaraju, "Fault-containing self-stabilizing algorithms", In Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, pp.45-54, 1996.

[9] S. Ghosh, A. Gupta and S.V. Pemmaraju, "A fault-containing self-stabilizing spanning tree algorithm", Journal of Computing and Information, Vol. 2(1), pp.322-338, 1996.

[10] S. Ghosh, A. Gupta and S.V. Pemmaraju, "Fault-containing network protocols", Symposium on Applied Computing archive Proceedings of the 1997 ACM symposium on Applied computing, pp.431-437, 1997.

[11] A. Gupta, "Fault-containing in self-stabilizing distributed algorithm", Ph.D. thesis, University of Iowa, 1997.

[12] S.T. Huang and N.S. Chen, "A self-stabilizing algorithm for constructing breadth-first trees", Information Processing Letters, Vol. 41, pp.107-117, 1992.

[13] T.C. Huang, “A self-Stabilizing algorithm for the shortest path problem assuming the distributed demon”, Computers and Mathematics with Applications, Vol. 50, pp.671-681, 2005.

[14] T.C. Huang, “A self-stabilizing algorithm for the shortest path problem assuming read/write atomicity”, Journal of Computer and System Sciences, Vol. 71, pp.70-85, 2005.

[15] T.C. Huang and J.C. Lin, “A self-stabilizing algorithm for the shortest path problem in a distributed system”, Computers and Mathematics with Applications, Vol.43, pp.103-109, 2002.

[16] T.C. Huang, J.C. Lin and J.N. Mou, “Self-stabilizing algorithms for the shortest path problem in a distributed system”, In: Proceedings of the ISCA 17th International Conference on the Parallel and Distributed Computing Systems, pp.270-277, 2004.

[17] J.C. Lin and T.C. Huang, “An efficient fault-containing self-stabilizing algorithm for finding a maximal independent set”, IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 8, pp.742-754, 2003.