JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 3, August 2006

A Self-Stabilizing Link-Coloring Protocol in Tree Networks with Permanent Byzantine Faults

Yusuke Sakurai*
Information and Communication Systems Group, Sharp Corporation

and
Fukuhito Ooshita† and Toshimitsu Masuzawa‡
Graduate School of Information Science and Technology, Osaka University

Self-stabilizing protocols can tolerate any type and any number of transient faults. But self-stabilizing protocols have no guarantee of their behavior against permanent faults. Thus, investigation concerning self-stabilizing protocols resilient to permanent faults is important. This paper proposes a self-stabilizing link-coloring protocol resilient to permanent Byzantine faults in tree networks. The protocol assumes the central daemon, and uses $\Delta + 1$ colors where Δ is the maximum degree in the network. This protocol guarantees that, for any nonfaulty process v, if the distance from v to any Byzantine ancestor of v is greater than two, v reaches its desired states within three rounds and never changes its states after that. Thus, it achieves fault containment with radius of two. Moreover, we prove that the containment radius becomes $\Omega(\log n)$ when we use only Δ colors, and prove that the containment radius becomes $\Omega(n)$ under the distributed daemon. These lower bound results prove necessity of $\Delta + 1$ colors and the central daemon to achieve fault containment with a constant radius.

I. Introduction

SELF-STABILIZATION[1] is one of the most effective and promising paradigms for fault-tolerant distributed computing.[2] A self-stabilizing protocol is guaranteed to achieve its desired behavior eventually regardless of the initial network configuration (i.e., global state). This implies a self-stabilizing protocol is resilient to any number and any type of transient faults since it can converge to its desired behavior from any configuration resulted by transient faults. However the convergence to the desired behavior is guaranteed only on the assumption that no further fault occurs during the convergence. Thus, a self-stabilizing protocol is not guaranteed to achieve its desired behavior in the presence of a permanent fault. Thus, it is strongly desired to design self-stabilizing protocols resilient to permanent faults.

There are some studies about self-stabilizing protocols resilient to permanent faults.[3–10] Most of these studies treat only crash faults, and these self-stabilizing protocols guarantee that each nonfaulty process achieves its desired behavior regardless of the initial network configuration. Nesterenko and Arora[9] treat Byzantine faults as permanent faults. The main difficulty in tolerating Byzantine faults is caused by arbitrary and unbounded state changes of the Byzantine processes: processes around the Byzantine processes may change their states in response to the state changes of the Byzantine processes, and processes next to the processes changing their states may also change

Received 01 September 2005; revision received 17 March 2006; accepted for publication 03 May 2006.

* Software Engineer, Information and Communication Systems Group, Sharp Corporation, Yamatokoriyama-shi, 639-1186 Japan
† Assistant Professor, Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560-8531 Japan; Tel. +81-6-6850-6583; Fax. +81-6-6850-6582; Email: f-oosita@ist.osaka-u.ac.jp
‡ Professor, Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560-8531 Japan

their states. This implies that the influence of the Byzantine processes expands to the whole system, and then no process can achieve its desired behavior. Nesterenko and Arora[9] give a novel definition of a self-stabilizing protocol resilient to Byzantine faults. The protocol guarantees, by containing the influence of Byzantine processes to only processes near them, that the other processes can achieve their desired behaviors eventually. They introduce the containment radius as the distance between a Byzantine process and processes affected by the Byzantine process. They also propose self-stabilizing protocols resilient to Byzantine faults for the vertex coloring problem and the dining philosophers problem. The containment radiuses are one for the vertex coloring problem and two for the dining philosophers problem.

The concept of fault containment is very popular in the field of self-stabilizing protocols.[11–16] However, these papers aim to contain the influence of a transient fault, and they do not consider (permanent) Byzantine faults.

In this paper, we consider a self-stabilizing link-coloring protocol resilient to Byzantine faults in tree networks. Link-coloring of a distributed system is an assignment of colors to the communication links such that no two communication links with the same color share a process in common. Link-coloring has many applications in distributed systems. For example, some network models (e.g., wireless networks or a one-port unidirectional model[17]) do not allow any process to engage multiple communications at the same time. In such models, we have to schedule communications between two processes under the restriction that each process can engage at most one communication at the same time. By assigning colors to links as above and scheduling communications so that only the links with the same color are used at the same time, we satisfy the restriction. Thus, link-coloring protocol is often used as the base protocol in such scheduling problems, e.g., data transfer scheduling and link scheduling in sensor networks. Therefore, many distributed protocols for link-coloring are proposed.[18–20] However, the fault tolerance is not considered in these protocols.

In this paper, we propose a self-stabilizing link-coloring protocol resilient to Byzantine faults. The protocol assumes the central daemon, i.e., exactly one process can execute an operation at each time, and uses $\Delta + 1$ colors, where Δ is the maximum degree of the network. The protocol guarantees that any nonfaulty process v reaches its desired states within three rounds and never changes its state after that if v has no Byzantine ancestor with the distance of two or less. Moreover, we show the following lower bound results for any self-stabilizing link-coloring protocol resilient to Byzantine faults:

1. When only Δ colors are used, the containment radius becomes $\Omega(\log n)$ if $\Delta \geq 3$, and $\Omega(n)$ if $\Delta = 2$, where n is the number of processes.
2. For any self-stabilizing link-coloring protocol that assumes the distributed daemon (i.e., an arbitrary number of processes can execute operations at each time), the containment radius is $\Omega(n)$ even when it can use arbitrary number of colors.

These two lower bound results prove necessity of $\Delta + 1$ colors and the central daemon to attain the fault containment against Byzantine faults with a constant containment radius.

II. Preliminaries

A. Distributed System

A *distributed system* $S = (P, L)$ consists of a set $P = \{v_1, v_2, \ldots, v_n\}$ of processes and a set L of communication links (simply called links). A link is an unordered pair of distinct processes, and processes v and w are called *neighbors* if $(v, w) \in L$. A distributed system S can be regarded as a graph with a vertex set P and a link set L, and thus, we use some graph terminologies for a distributed system S.

We consider *rooted tree networks* in this paper. For each process $v \in P$, N_v denotes the set of neighbors of v, prt_v denotes the parent of v, and Ch_v denotes the set of children of v. We do not assume existence of a unique identifier of each process. Instead we assume each process can identify its parent from its neighbors, and distinguish its children by having a local order. The x-th child of process v is denoted by $ch_v(x)$ $(1 \leq x \leq |Ch_v|)$. The distance from the root process to process v is called the *depth* of v. The maximum degree of a tree network is denoted by Δ, i.e., the root process has at most Δ children and any other process has at most $\Delta - 1$ children.

Each process is modeled by a state machine that can communicate with its neighbors through link registers. For each pair of neighboring processes, u and v, there are two link registers $r_{u,v}$ and $r_{v,u}$. Message transmission from u to v is realized as follows: u writes a message to link register $r_{u,v}$ and then v reads it from $r_{u,v}$.

For each process v, let $In_v = \{r_{u,v} | u \in N_v\}$ be the input register set of v and $Out_v = \{r_{v,u} | u \in N_v\}$ be the output register set of v. For convenience, we use variables to denote the states of a process and a link register, and use *guarded actions* (simply called *actions*) to denote the state transition function of a process. Each action is of the following form:

$$\langle guard \rangle \longrightarrow \langle statement \rangle$$

The guard of an action of a process v is a boolean expression on the variables of v and all input registers r ($r \in In_v$). The statement of an action of v updates one or more variables of v and all output registers r ($r \in Out_v$). The values assigned to the variables of v and its output registers r ($r \in Out_v$) depend only on the values of variables of v and its input registers r ($r \in In_v$). The statement of an action can be executed only if its guard evaluates to true. When guards of multiple actions evaluate to true, one of these actions is deterministically selected and executed. We assume that each action is atomically executed: the evaluation of the guard and the execution of the corresponding statement of the action, if executed, are done in one atomic step. The execution of an action of v is called a *step* of v.

A global state of a distributed system is called a *configuration* and is denoted by a conjunction of states of all processes and all link registers. We define C as the set of all possible configurations of a distributed system S. For each configuration $\rho \in C$, $\rho|v$ and $\rho|r$ denote the states of process v and link register r in configuration ρ respectively.

When a process v has a guarded action whose guard is true at configuration ρ, we say v is *enabled* at ρ. Let $En(\rho, v)$ be a predicate such that $En(\rho, v) = true$ iff v is enabled at ρ. Letting Q be any set of processes, when configuration ρ changes to configuration ρ' by executing actions of every enabled process in Q, we denote $\rho \overset{Q}{\mapsto} \rho'$.

A *schedule* of a distributed system is an infinite sequence of sets of processes. Let $\mathcal{Q} = Q_1, Q_2, \ldots$ be a schedule. An infinite sequence of configurations $e = \rho_0, \rho_1, \ldots$ is called an *execution* from an initial configuration ρ_0 by a schedule $\mathcal{Q}$, if e satisfies $\rho_i \overset{Q_{i+1}}{\mapsto} \rho_{i+1}$ for each i ($i \geq 0$). Notice that the execution e is uniquely determined from its initial configuration and a schedule $\mathcal{Q}$ since the executed action of each process is deterministically selected (even when a process has two or more actions with true guards). The set of possible schedules in a distributed system is sometimes modeled by a scheduler called a daemon. In this paper, we consider two kinds of daemons, the *distributed daemon* and the *central daemon*. Under the distributed daemon, each Q_i can be an arbitrary set of processes. That is, the distributed daemon allows two or more processes to execute their actions simultaneously. By contrast, the central daemon is a special case of the distributed daemon. Under the central daemon, $|Q_i| = 1$ holds for each i, i.e., no two processes execute their actions simultaneously. Under the central daemon, when $Q_i = \{q_i\}$ for each i, we simply describe a schedule as $\mathcal{Q} = q_1, q_2, \ldots$ and describe a configuration transition as $\rho_i \overset{q_{i+1}}{\mapsto} \rho_{i+1}$ instead of $\rho_i \overset{Q_{i+1}}{\mapsto} \rho_{i+1}$. The set of all possible executions from an initial configuration $\rho_0 \in C$ is denoted by E_{ρ_0}. The set of all possible executions is denoted by E, that is, $E = \bigcup_{\rho_0 \in C} E_{\rho_0}$.

We consider *asynchronous* distributed systems where we can make no assumption on schedules except that any schedule is *weakly fair*: every process appears in the schedule infinitely often. Notice that the weakly fair schedules do not necessarily imply that processes actually execute actions infinitely often. If no guards of a process evaluate to true whenever the process is activated, the process cannot execute any action.

In this paper, we consider two kinds of permanent faults: crash faults and Byzantine faults.

- *crash faults*: A crashed process (i.e., a process with the crash fault) prematurely stops execution of its actions. If v is a crashed process, v does not change states of v and its output registers r ($r \in Out_v$) after certain time even when there is an action with true guard. Before that time, v acts as a nonfaulty process and correctly executes its actions.
- *Byzantine faults*: A Byzantine process (i.e., a process with the Byzantine fault) can arbitrarily behave independently from its actions. If v is a Byzantine process, v can repeatedly change states of v and its output registers r ($r \in Out_v$) arbitrarily.

We define BF and CF as the sets of Byzantine processes and crashed processes respectively. Since a crash fault can be regarded as a special case of the Byzantine fault, $BF \supseteq CF$ holds. However, in what follows, we assume without loss of generality that $BF \cap CF = \emptyset$ holds by excluding crashed processes from the set BF.

Let $CF = \{f_1, f_2, \ldots, f_c\}$. In distributed systems where faults can occur, an infinite sequence of configurations $e = \rho_0, \rho_1, \ldots$ is called an *execution* by a schedule $\mathcal{Q} = Q_1, Q_2, \ldots$, if there exists $t_1, t_2, \ldots, t_c$ such that the following conditions hold for any i ($i \geq 0$):

- For any $v \in Q_{i+1} - (BF \cup CF)$, execution of an action of v changes v's state from $\rho_i|v$ to $\rho_{i+1}|v$ (possibly $\rho_i|v = \rho_{i+1}|v$) and changes the state of every output register r ($r \in Out_v$) from $\rho_i|r$ to $\rho_{i+1}|r$ (possibly $\rho_i|r = \rho_{i+1}|r$).
- For any $f_j \in Q_{i+1} \cap CF$, if $i \geq t_j$, the states of f_j and each output register r ($r \in Out_{f_j}$) remain unchanged from ρ_i to ρ_{i+1}: $\rho_i|f_j = \rho_{i+1}|f_j$ and $\forall r \in Out_{f_j} : \rho_i|r = \rho_{i+1}|r$ hold. If $i < t_j$, execution of an action of f_j changes its state from $\rho_i|f_j$ to $\rho_{i+1}|f_j$ (possibly $\rho_i|f_j = \rho_{i+1}|f_j$) and changes the state of every output register r ($r \in Out_{f_j}$) from $\rho_i|r$ to $\rho_{i+1}|r$ (possibly $\rho_i|r = \rho_{i+1}|r$). Notice that t_j implies that process f_j becomes crashed between ρ_{t_j-1} and ρ_{t_j}.
- For any $v \notin Q_{i+1}$, $\rho_i|v = \rho_{i+1}|v$ and $\forall r \in Out_v : \rho_i|r = \rho_{i+1}|r$ hold.

Notice that, for any process $v \in Q_{i+1} \cap BF$, $\rho_{i+1}|v$ and $\rho_{i+1}|r$ ($r \in Out_v$) can be arbitrary states.

In asynchronous distributed systems, time is usually measured by *asynchronous rounds* (simply called *rounds*). Let $e = \rho_0, \rho_1, \ldots$ be an execution from configuration ρ_0 by a schedule $\mathcal{Q} = Q_1, Q_2, \ldots$. The first round of e is defined to be the minimum prefix of e, $e' = \rho_0, \rho_1, \ldots, \rho_k$, such that $\bigcup_{i=1}^{k} Q_i = P$. Round t ($t \geq 2$) is defined recursively, by applying the above definition of the first round to $e'' = \rho_k, \rho_{k+1}, \ldots$. Intuitively, every process has a chance to update its state in every round.

In Appendix A, we summarize notations in Table 1.

B. Self-Stabilizing Protocol Resilient to Byzantine Faults

In this paper, we treat only *static problems*, i.e., once the system reaches a desired configuration, the configuration remains unchanged forever. For example, the spanning-tree construction problem, the leader election problem and the coloring problem are examples of static problems, but the mutual exclusion problem is not a static problem[2]. A static problem can be defined by a *specification predicate*, $spec(v)$, for each process v, which specifies the condition that v should satisfy at the desired configuration. A specification predicate $spec(v)$ is a boolean expression on the variables of processes $P_v \subseteq P$ and link registers $R_v \subseteq R$, where R is the set of all link registers.

A self-stabilizing protocol is a protocol that guarantees each process v satisfies $spec(v)$ eventually regardless of the initial configuration. By this property, a self-stabilizing protocol can tolerate any number and any type of transient faults. However, since we consider permanent faults such as Byzantine faults and crash faults, faulty processes cannot satisfy $spec(v)$. In addition, nonfaulty processes near the faulty processes can be influenced by the faulty processes and cannot satisfy $spec(v)$. Thus, Nesterenko and Arora[9] define a self-stabilizing protocol resilient to these faults. Informally, the protocol requires each nonfaulty process v far from any faulty process to satisfy $spec(v)$ eventually. They also propose concepts of *strict tolerance* and *strict stabilization* to define some classes of protocols resilient to Byzantine faults. We combine the above two concepts, and propose $(\mathcal{B}, \mathcal{C})$-*self-stabilization with containment radius* (τ, μ), where $\mathcal{B}$ and $\mathcal{C}$ represent Byzantine faults and crash faults respectively. In the following definition, let $\Gamma(v, l)$ be the set of processes whose distances to v are l or less.

Definition 1. A configuration ρ_0 is a $(\mathcal{B}, \mathcal{C})$-*stable configuration with containment radius* (τ, μ) if and only if, for any execution $e = \rho_0, \rho_1, \ldots$ and any process v, the following condition holds:

If the distance from v to any Byzantine process is more than τ (i.e., $\forall w \in \Gamma(v, \tau) : w \notin BF$) and the distance from v to any crashed process is more than μ (i.e., $\forall u \in \Gamma(v, \mu) : u \notin CF$), for any t ($t \geq 0$),

i) v satisfies $spec(v)$ in ρ_t,

ii) both $\rho_t|v = \rho_{t+1}|v$ and $\forall r \in Out_v : \rho_t|r = \rho_{t+1}|r$ hold.

Definition 1 states, once the system reaches a stable configuration, a process v sufficiently far from any faulty process always satisfies $spec(v)$ and never changes the states of v and its output registers r ($r \in Out_v$) forever.

Definition 2. A protocol A is a $(\mathcal{B}, \mathcal{C})$-*self-stabilizing protocol with containment radius* (τ, μ) if and only if, for any execution $e = \rho_0, \rho_1, \ldots$ of A starting from any configuration ρ_0, there exists ρ_t that is a $(\mathcal{B}, \mathcal{C})$-stable configuration with containment radius (τ, μ). We say that the *stabilization time* of A is k rounds for the minimum k such that the last configuration of the k-th round is a stable configuration in any execution of a protocol A.

Definition 2 states a $(\mathcal{B}, \mathcal{C})$-self-stabilizing protocol guarantees that the system eventually reaches a $(\mathcal{B}, \mathcal{C})$-stable configuration from any initial configuration. If a protocol A is a $(\mathcal{B}, \mathcal{C})$-self-stabilizing protocol with containment radius (τ, μ) for some constant τ and μ, A is strictly $\mathcal{C}$-tolerant[9] and strictly stabilizing.[9]

C. Link-Coloring Problem

A link-coloring problem is to find an assignment of colors to links such that no two links with the same color share a process in common. In the following, let $CSET$ be a given set of colors, and let $Color(e) \in CSET$ be a color of link e. We define the *distributed link-coloring problem* as follows.

Definition 3. In the distributed link-coloring problem, the specification predicate $spec(v)$ for a process v is given as follows:

$$\forall x, y \in N_v : x \neq y \Longrightarrow Color((v, x)) \neq Color((v, y))$$

In the following, we define a *b-link-coloring protocol* as a link-coloring protocol using b colors.

III. Link-Coloring Protocol under the Central Daemon

In this section, we propose a $(\mathcal{B}, \mathcal{C})$-self-stabilizing $(\Delta + 1)$-link-coloring protocol with containment radius (2, 1). Our protocol uses at most $\Delta + 1$ colors for link-coloring, and thus, we assume $CSET = \{1, 2, \ldots, \Delta + 1\}$.

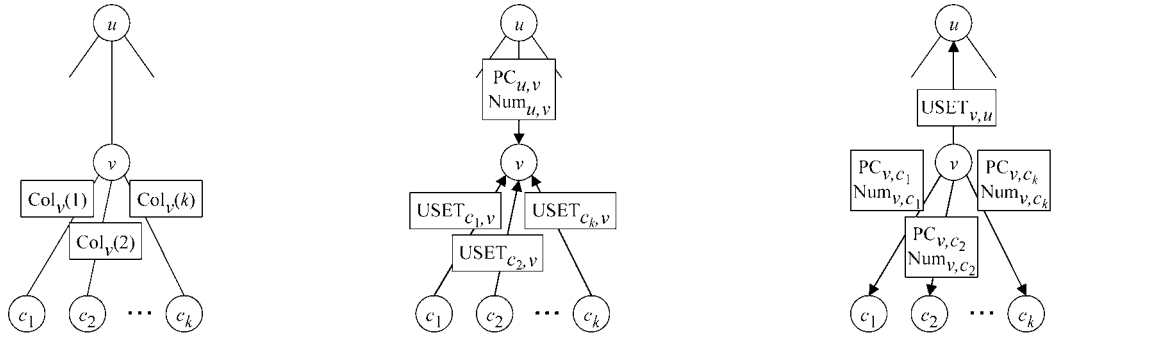
Let v be any process, $u = prt_v$, and x_v be an integer satisfying $v = ch_u(x_v)$, that is, v is the x_v-th child of u. First, we explain variables of a process and a link register (See Fig. 1).

- Process v has variables $\texttt{Col}_v(x)$ $(1 \leq x \leq |Ch_v|)$. Variable $\texttt{Col}_v(x)$ denotes a color of link $(v, ch_v(x))$. Notice that v does not have the variable to store the color of link (u, v) for its parent u. The color of the link is stored in $\texttt{Col}_u(x_v)$ of u.
- Link register $r_{u,v}$ has variables $\texttt{Num}_{u,v}$ and $\texttt{PC}_{u,v}$. Process u assigns x_v to $\texttt{Num}_{u,v}$, and assigns $\texttt{Col}_u(x_v)$ to $\texttt{PC}_{u,v}$. Process v can learn the color of link (u, v) by reading $\texttt{PC}_{u,v}$. The value of $\texttt{Num}_{u,v}$ is used to determine $\texttt{Col}_v(x)$ $(1 \leq x \leq |Ch_v|)$.
- Link register $r_{v,u}$ has a variable $\texttt{USET}_{v,u}$. Process v assigns $\{\texttt{Col}_v(x) \mid 1 \leq x \leq |Ch_v|\}$ to $\texttt{USET}_{v,u}$. Process u can learn the colors assigned to links $(v, ch_v(x))$ $(1 \leq x \leq |Ch_v|)$ by reading $\texttt{USET}_{v,u}$.

For simplicity, we assume that $\texttt{Col}_v(x) \in CSET$ $(1 \leq x \leq |Ch_v|)$, $1 \leq \texttt{Num}_{u,v} \leq \Delta$, $\texttt{PC}_{u,v} \in CSET$, and $\texttt{USET}_{v,u} \subseteq CSET$ are always satisfied at any configuration even when there exist some Byzantine processes. Notice that we can make these assumptions without loss of generality, because these assumptions impose restriction only on the domains of the variables. For example, when the variable $\texttt{Col}_v(x)$ is a variable of integer type, we can regard $(\texttt{Col}_v(x) \bmod (\Delta + 1)) + 1$ as its actual value so that $\texttt{Col}_v(x) \in CSET$ should hold. In Appendix A, we summarize these notations in Table 2.

Process v executes the following steps atomically:

1. Process v reads variables on all link registers in In_v.
2. Process v locally determines colors $\texttt{Col}_v(x)$ for all x $(1 \leq x \leq |Ch_v|)$.



(a) Variables of v. (b) Variables of registers in In_v. (c) Variables of registers in Out_v.

Fig. 1 Variables of v, link registers in In_v, and link registers in Out_v, where $u = prt_v$ and $c_x = ch_v(x)$ $(1 \leq x \leq |Ch_v|)$.

3. For each x ($1 \leq x \leq |Ch_v|$), letting $w = ch_v(x)$, process v writes x and $\texttt{Col}_v(x)$ to $\texttt{Num}_{v,w}$ and $\texttt{PC}_{v,w}$ on link register $r_{v,w}$ respectively.
4. Letting $u = prt_v$, process v writes $\{\texttt{Col}_v(x) | 1 \leq x \leq |Ch_v|\}$ to $\texttt{USET}_{v,u}$ on link register $r_{v,u}$.

We show the protocol LINKCOLORING in Fig. 2. For simplicity, we show the protocol by the pseudo-code instead of guarded actions. The function LINKCOLORING is the pseudo-code representing a guarded action, and is executed in one atomic step. Notice that we can transform this protocol into a guarded action as follows: the statement of the action of process v is the function LINKCOLORING for v, and the guard of the action is the boolean function f such that f is true in configuration ρ if and only if execution of the function LINKCOLORING for v in ρ changes some variables of v and its output registers r ($r \in Out_v$).

To explain how each process v determines $\texttt{Col}_v(x)$, we define candidate color sets and preference colors. For each process v and each x ($1 \leq x \leq |Ch_v|$), we define the *candidate color set* $CCol_v(x)$ as follows:

$$CCol_v(x) = \{x, x+1, x+2\}$$

For each non-root process v and each x ($1 \leq x \leq |Ch_v|, x \neq \texttt{Num}_{prt_v,v}$), we define the *preference color* $PCol_v(x)$ as follows:

$$PCol_v(x) = \begin{cases} x & (x < \texttt{Num}_{prt_v,v}) \\ x+2 & (x > \texttt{Num}_{prt_v,v}) \end{cases}$$

We show an example of $CCol_v(x)$ and $PCol_v(x)$ in Fig. 3. The set described on each link denotes the candidate color set of the link, and the underlined integer denotes the preference color.

In the protocol LINKCOLORING, nonfaulty process v assigns colors to links with the following policies: 1) Process v always assigns colors to links so that no two links connecting to v has the same color, 2) Process v always assigns color $c \in CCol_v(x)$ to $\texttt{Col}_v(x)$, 3) If possible, v assigns color $PCol_v(x)$ to $\texttt{Col}_v(x)$, and 4) If possible, v assigns a color to $\texttt{Col}_v(\texttt{Num}_{u,v})$ such that no two links connecting to $ch_v(\texttt{Num}_{u,v})$ have the same color. In the case that v is the root process, v assigns x to $\texttt{Col}_v(x)$ ($1 \leq x \leq |Ch_v|$) (See lines 4 to 6). In the following paragraph, we explain the assignment of v in the case that v is not the root process.

First, v assigns a color to $\texttt{Col}_v(x)$ for each x ($x \neq \texttt{Num}_{u,v}$).

```
1:  function LinkColoring {
2:    // v is the root process
3:    if v = root then
4:      for each x (1 ≤ x ≤ |Ch_v|) {
5:        Col_v(x) := x
6:      }
7:
8:    // v is not the root process
9:    else
10:     u := prt_v
11:
12:     // assign a color to Col_v(x) (x ≤ Num_{u,v} − 1)
13:     for each x (1 ≤ x ≤ min{Num_{u,v} − 1, |Ch_v|}) {
14:       if x < PC_{u,v} then Col_v(x) = x endif
15:       if x ≥ PC_{u,v} then Col_v(x) = x + 1 endif
16:     }
17:
18:     // assign a color to Col_v(x) (Num_{u,v} + 1 ≤ x)
19:     for each x (Num_{u,v} + 1 ≤ x ≤ |Ch_v|) {
20:       if x > PC_{u,v} − 2 then Col_v(x) = x + 2 endif
21:       if x ≤ PC_{u,v} − 2 then Col_v(x) = x + 1 endif
22:     }
23:     // assign a color to Col_v(x) (x = Num_{u,v})
24:     if |Ch_v| ≥ Num_{u,v} then
25:       x := Num_{u,v}
26:       // candidate color set of Col_v(x)
27:       CCOL := {x, x + 1, x + 2}
28:       if Col_v(x) ∉ CCOL or
29:            PC_{u,v} = Col_v(x) or
30:            ∃x'(x' ≠ x) : Col_v(x') = Col_v(x) then
31:         CCOL := CCOL − {PC_{u,v}} − {Col_v(x') | x' ≠ x}
32:         C := CCOL − USET_{ch_v(x),v}
33:         if C ≠ ∅ then Col_v(x) := min(C) endif
34:         if C = ∅ then Col_v(x) := min(CCOL) endif
35:       endif
36:     endif
37:   endif
38:
39:   // write colors to link registers
40:   for each x (1 ≤ x ≤ |Ch_v|) {
41:     Num_{v,ch_v(x)} := x
42:     PC_{v,ch_v(x)} := Col_v(x)
43:   }
44:   if v ≠ root then
45:     USET_{v,u} := { Col_v(x) | 1 ≤ x ≤ |Ch_v| }
46:   endif
47: }
```

Fig. 2 The protocol LINKCOLORING: the action of v.

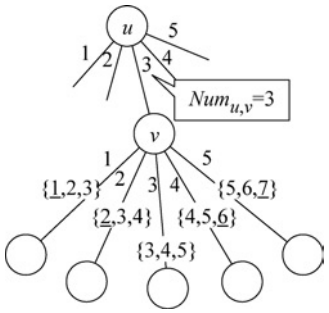


Fig. 3 The candidate color set and the preference color, where $u = prt_v$.

- For each x ($1 \leq x \leq \min\{\mathtt{Num}_{u,v} - 1, |Ch_v|\}$), v assigns a color as follows (See line 12 to 16):

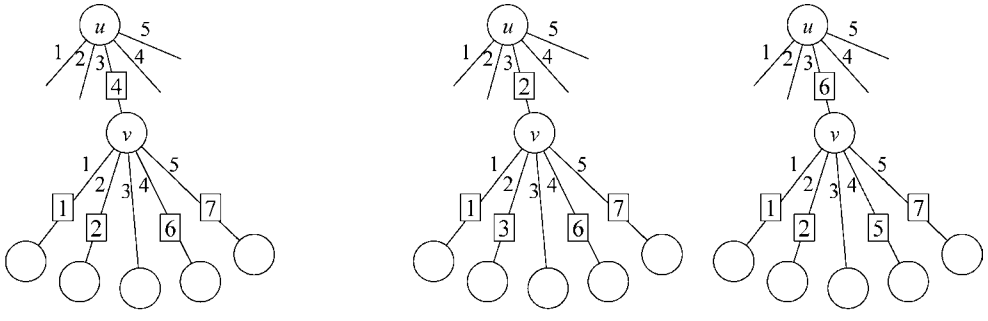
$$\mathtt{Col}_v(x) = \begin{cases} x & (x < \mathtt{PC}_{u,v}) \\ x+1 & (x \geq \mathtt{PC}_{u,v}) \end{cases}$$

- For each x ($\mathtt{Num}_{u,v} + 1 \leq x \leq |Ch_v|$), v assigns a color as follows (See line 18 to 22):

$$\mathtt{Col}_v(x) = \begin{cases} x+2 & (x > \mathtt{PC}_{u,v} - 2) \\ x+1 & (x \leq \mathtt{PC}_{u,v} - 2) \end{cases}$$

In this assignment, if $\mathtt{PC}_{u,v} \in \{\mathtt{Num}_{u,v}, \mathtt{Num}_{u,v} + 1, \mathtt{Num}_{u,v} + 2\}$ holds, v assigns $PCol_v(x)$ to $\mathtt{Col}_v(x)$. We show an example in Fig. 4(a). The boxed integer on each link denotes the color of the link. If either $\mathtt{PC}_{u,v} < \mathtt{Num}_{u,v}$ or $\mathtt{PC}_{u,v} > \mathtt{Num}_{u,v} + 2$ holds, v assigns a color in $CCol_v(x)$ to $\mathtt{Col}_v(x)$ (See Fig. 4(b) and (c)).

Next, v determines a color of $\mathtt{Col}_v(x)$ in the case that $x = \mathtt{Num}_{u,v}$ (See lines 23 to 36). Notice that this case happens if $|Ch_v| < \mathtt{Num}_{u,v}$ holds (See line 24). In this case, v changes $\mathtt{Col}_v(x)$ only if v has to change it, that is, either $\mathtt{Col}_v(x) \notin CCol_v(x)$ holds or two links connecting to v have the same color (See lines 28 to 30). Variable $CCOL$ is used as the candidate color of $\mathtt{Col}_v(x)$, and $CCol_v(x)$ is assigned to $CCOL$ on line 27. On lines 31 to 34, v determines the new value of $\mathtt{Col}_v(x)$. On line 31, v reduces colors used around v from $CCOL$. Notice that $CCOL$ has at least two colors after this reduction. On line 32, v makes color set C by reducing colors used around $ch_v(x)$ from $CCOL$. If $C \neq \emptyset$ holds, v assigns any color $c \in C$ to $\mathtt{Col}_v(x)$ (e.g., $c = \min(C)$ on line 33). Otherwise, v assigns any color $c \in CCOL$ (e.g., $c = \min(CCOL)$ on line 34). This assignment is based on the above policy 4. We show two typical examples in Fig. 5. The value in a wide box denotes the value assigned to $\mathtt{Col}_v(x)$. In the case of Fig. 5(a), v can assign a color so that no two links with the same color share v or $ch_v(x)$. In the case of Fig. 5(b), v cannot do so, and assigns a color so that no two links with the same color share v.



(a) $\mathtt{PC}_{u,v} \in \{\mathtt{Num}_{u,v}, \mathtt{Num}_{u,v} + 1, \mathtt{Num}_{u,v} + 2, \}$ (b) $\mathtt{PC}_{u,v} < \mathtt{Num}_{u,v}$ (c) $\mathtt{PC}_{u,v} > \mathtt{Num}_{u,v} + 2$

Fig. 4 The assignment of $\mathtt{Col}_v(x)$ for x ($x \neq \mathtt{Num}_{u,v}$).

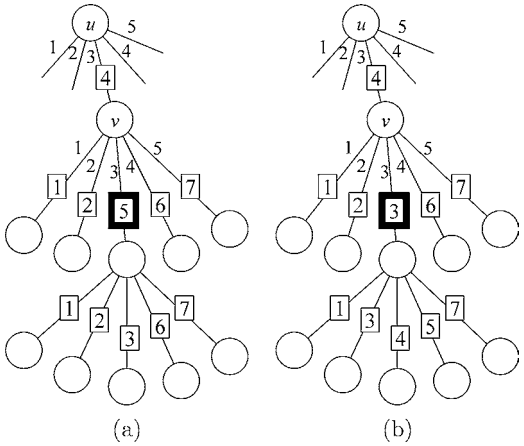


Fig. 5 The assignment of $\mathtt{Col}_v(x)$ for $x = \mathtt{Num}_{u,v}$.

After v executes an action, the guard of the action of v becomes false and $spec(v)$ becomes true. Notice that, even if v's child w changes $\mathtt{USET}_{w,v}$, v does not change any variable. This is because the guard of the action of v does not include variables $\mathtt{USET}_{w,v}$. In addition, $spec(v)$ does not include variables $\mathtt{USET}_{w,v}$ ($w \in Ch_v$). Thus, once v executes an action, even when $w \in Ch_v$ is a Byzantine process and changes $\mathtt{USET}_{w,v}$ arbitrarily, the guard of the action of v remains false and $spec(v)$ remains true unless prt_v changes $\mathtt{PC}_{prt_v,v}$.

Assume v and $u = prt_v$ are nonfaulty processes, and prt_u is a Byzantine process. In what follows, we explain how the influence of the Byzantine process prt_u is contained in the candidate color sets and the preference colors.

Since u is a nonfaulty process, by the protocol, u assigns a color in $CCol_u(x)$ to $\mathtt{Col}_u(x)$ in the first round. Thus, process u reduces the influence of Byzantine process prt_u in the sense that the change of $\mathtt{Col}_u(x)$ is constrained in $CCol_u(x)$ while the change of $\mathtt{PC}_{prt_u,u}$ is completely unconstrained.

As described above, u always assigns a color in $CCol_u(x)$ to $\mathtt{Col}_u(x)$ ($1 \leq x \leq |Ch_u|$). Consequently, letting $C = \{\mathtt{Num}_{u,v}, \mathtt{Num}_{u,v}+1, \mathtt{Num}_{u,v}+2\}$, $\mathtt{PC}_{u,v} = \mathtt{Col}_u(\mathtt{Num}_{u,v}) \in CCol_u(\mathtt{Num}_{u,v}) = C$ holds at any configuration after the first round. Then, when v executes a step after the first round, v assigns $PCol_v(x)$ to $\mathtt{Col}_v(x)$ for each x ($x \neq \mathtt{Num}_{u,v}$), and assigns a color in $CCol_v(\mathtt{Num}_{u,v})$ to $\mathtt{Col}_v(\mathtt{Num}_{u,v})$. This implies v never changes $\mathtt{Col}_v(x)$ ($x \neq \mathtt{Num}_{u,v}$) even when u changes $\mathtt{PC}_{u,v}$. However, v may have to change $\mathtt{Col}_v(\mathtt{Num}_{u,v})$ in response to change of $\mathtt{PC}_{u,v}$. Let $w = ch_v(\mathtt{Num}_{u,v})$. Since w also assigns $PCol_w(x)$ to $\mathtt{Col}_w(x)$ for each x ($x \neq \mathtt{Num}_{v,w}$) after the second round, links that colors in C can be assigned to and connects to either v or w are only (u, v), (v, w), and $(w, ch_w(\mathtt{Num}_{u,v}))$. Thus, v can assign a color in C to $\mathtt{Col}_v(\mathtt{Num}_{u,v})$ so that the color of (v, w) can differ from those of (u, v) and $(w, ch_w(\mathtt{Num}_{u,v}))$. Therefore, w, a process apart from the Byzantine ancestor by distance of three, is not affected by the Byzantine process, and we attain the fault containment against the Byzantine faults. Concerning the protocol LINKCOLORING, we have the following theorem.

Theorem 1. The protocol LINKCOLORING is a $(\Delta + 1)$-link-coloring protocol satisfying the following property:

- Let $e = \rho_0, \rho_1, \ldots$ be any execution, $v \in P$ be any nonfaulty process, and ρ_s be the last configuration of the third round. When v has no Byzantine ancestor with the distance of two or less and has no crashed parent, for any t ($t \geq s$),
  - i) v satisfies $spec(v)$ in ρ_t, and,
  - ii) both $\rho_t|v = \rho_{t+1}|v$ and $\forall r \in Out_v : \rho_t|r = \rho_{t+1}|r$ hold. □

From Theorem 1, we clearly have the following corollary.

Corollary 1. The protocol LINKCOLORING ia a $(\mathcal{B}, \mathcal{C})$-self-stabilizing $(\Delta + 1)$-link-coloring protocol with containment radius (2, 1). The stabilization time of LINKCOLORING is three rounds. □

In what follows, we formally prove Theorem 1. First, we redefine the specification predicate $spec(v)$ with the variables used in the protocol LINKCOLORING.

- If v is the root process,

$$spec(v) = \forall x, y(1 \leq x < y \leq |Ch_v|) : \mathtt{Col}_v(x) \neq \mathtt{Col}_v(y).$$

- If v is not the root process, letting x_v be an integer satisfying $v = ch_{prt_v}(x_v)$,

$$spec(v) = [\forall x, y(1 \leq x < y \leq |Ch_v|) : \mathtt{Col}_v(x) \neq \mathtt{Col}_v(y)]$$
$$\wedge [\forall x(1 \leq x \leq |Ch_v|) : \mathtt{Col}_v(x) \neq \mathtt{Col}_{prt_v}(x_v)]$$

Notice that, for each leaf v, $spec(v)$ always holds. Thus, in the following, we consider only the root process and intermediate processes.

Next, we define three states of a process v.

Definition 4. Process v is in a *candidate state* if v satisfies the following two conditions:
- $\forall x, y(1 \leq x < y \leq |Ch_v|) : \mathtt{Col}_v(x) \neq \mathtt{Col}_v(y)$.
- $\forall x(1 \leq x \leq |Ch_v|) : \mathtt{Col}_v(x) \in CCol_v(x)$.

Definition 5. Process v is in a *preference state* if v satisfies the following two conditions:
- Process v is in a candidate state.
- If v is not the root process, $\forall x(1 \leq x \leq |Ch_v|, x \neq \mathtt{Num}_{prt_v,v}) : \mathtt{Col}_v(x) = PCol_v(x)$ holds.

Definition 6. Process v is in a *stable state* if v satisfies the following two conditions:
- Process v is in a preference state.
- If v is not the root process, letting x_v be an integer satisfying $v = ch_{prt_v}(x_v)$, $\forall x(1 \leq x \leq |Ch_v|) : \mathtt{Col}_v(x) \neq \mathtt{Col}_{prt_v}(x_v)$ holds.

Notice that, for the root process r, if r is in a candidate state, r is also in a preference state and in a stable state, and thus, the above three states are equivalent for the root process r. In the following, we use predicates $CS(\rho, v)$, $PS(\rho, v)$, and $SS(\rho, v)$ to state that v is in a candidate state, in a preference state, and in a stable state in configuration ρ respectively. Notice that, if $SS(\rho, v)$ holds, v satisfies $spec(v)$ in ρ, and $\neg En(\rho, v)$ holds.

Recall that nonfaulty process v satisfies $spec(v)$ immediately after executing an action, and the guards of actions of v and $spec(v)$ do not include variables $\mathtt{USET}_{w,v}$ ($w \in Ch_v$). Thus, once v executes an action, even when $w \in Ch_v$ is a Byzantine process and changes $\mathtt{USET}_{w,v}$ arbitrarily, the guards of actions of v remain false and $spec(v)$ remains true unless prt_v changes $\mathtt{PC}_{prt_v,v}$.

We consider two cases:

A. Processes v, prt_v, and prt_{prt_v} are nonfaulty processes (Lemma 3),

B. Process v and prt_v are nonfaulty processes, and prt_{prt_v} is a crashed process (Lemma 4).

First, we consider the case A.

Lemma 1. Let $\mathcal{Q} = q_1, q_2, \ldots$ be a schedule, and $e = \rho_0, \rho_1, \ldots$ be an execution by $\mathcal{Q}$. Let v be a nonfaulty process and ρ_s be the last configuration of the first round. Then, $CS(\rho_t, v)$ holds in any configuration ρ_t ($t \geq s$).

Proof. By the definition of the round, there exists l such that $q_l = v$ ($l \leq s$). By the protocol, it is obvious that $CS(\rho_l, v)$ holds. For any i ($i \geq l$), since $CS(\rho_i, v)$ depends on only the variables of v in ρ_i, $CS(\rho_i, v)$ still holds even when any other process executes a step. In addition, $CS(\rho_i, v)$ still holds even when v re-executes a step. Thus, the lemma holds. □

Lemma 2. Let $\mathcal{Q} = q_1, q_2, \ldots$ be a schedule, and $e = \rho_0, \rho_1, \ldots$ be an execution by $\mathcal{Q}$. Let v be a process such that v is not the root and both v and prt_v are nonfaulty. Let ρ_s be the last configuration of the second round. Then, $PS(\rho_t, v)$ holds in any configuration ρ_t ($t \geq s$).

Proof. Let $u = prt_v$, x_v be an integer satisfying $v = ch_u(x_v)$, and ρ_k be the last configuration of the first round. Then, for any ρ_j ($j \geq k$), $CS(\rho_j, u)$ holds by Lemma 1. Since u is a nonfaulty process, by the protocol, both $\mathtt{PC}_{u,v} = \mathtt{Col}_u(x_v)$ and $\mathtt{Num}_{u,v} = x_v$ hold in ρ_j. Thus, $\mathtt{PC}_{u,v} = \mathtt{Col}_u(\mathtt{Num}_{u,v}) \in \{\mathtt{Num}_{u,v}, \mathtt{Num}_{u,v} + 1, \mathtt{Num}_{u,v} + 2\}$ holds

in ρ_j. By the definition of the round, there exists l such that $v = q_l$ ($k < l \leq s$). Then, since $\mathtt{PC}_{u,v} \in CCol_u(\mathtt{Num}_{u,v})$, v assigns $PCol_v(x)$ to $\mathtt{Col}_v(x)$ for each x ($x \neq \mathtt{Num}_{u,v}$), and thus, $PS(\rho_l, v)$ holds. For any i ($i \geq l$), since $PS(\rho_i, v)$ depends on only the variables of v in ρ_i, $PS(\rho_i, v)$ still holds even when any other process executes a step. In addition, $PS(\rho_i, v)$ still holds even when v re-executes a step. Thus, the lemma holds. □

Lemma 3. Let $\mathcal{Q} = q_1, q_2, \ldots$ be a schedule, and $e = \rho_0, \rho_1, \ldots$ be an execution by $\mathcal{Q}$. Let v be a process satisfying one of the following conditions:

1. Process v is the root process and is nonfaulty.
2. Process v is a child of the root process, and both v and prt_v are nonfaulty.
3. The depth of v is two or more, and all of v, prt_v, and prt_{prt_v} are nonfaulty.

Then, letting ρ_s be the last configuration of the third round, for any t ($t \geq s$),

i) v satisfies $spec(v)$ in ρ_t, and,
ii) both $\rho_t|v = \rho_{t+1}|v$ and $\forall r \in Out_v : \rho_t|r = \rho_{t+1}|r$ hold.

Proof. Case that v satisfies the condition 1. By the protocol, once v executes a step in the first round, v satisfies $spec(v)$ and never changes its state after that. Thus, the lemma holds.

Case that v satisfies the condition 2. By the protocol, once prt_v executes a step in the first round, prt_v never changes $\mathtt{PC}_{prt_v,v}$ after that. Then, once v executes a step in the second round, v satisfies $spec(v)$ and never changes its state after that. Thus, the lemma holds.

Case that v satisfies the condition 3. Let $u = prt_v$, x_v be an integer satisfying $v = ch_u(x_v)$, and ρ_k be the last configuration of the second round. By Lemma 2, for any j ($j \geq k$), $PS(\rho_j, u)$ and $PS(\rho_j, v)$ hold. Notice that, since u is a nonfaulty process, by the protocol, both $\mathtt{PC}_{u,v} = \mathtt{Col}_u(x_v)$ and $\mathtt{Num}_{u,v} = x_v$ hold in ρ_j. We consider two cases.

- Case that $\mathtt{Num}_{u,v} \neq \mathtt{Num}_{prt_u,u}$ holds. Since $PS(\rho_j, u)$ and $\mathtt{Num}_{u,v} \neq \mathtt{Num}_{prt_u,u}$ hold, $\mathtt{PC}_{u,v} = \mathtt{Col}_u(\mathtt{Num}_{u,v}) = PCol_u(\mathtt{Num}_{u,v})$ holds in ρ_j, and thus, $\mathtt{PC}_{u,v}$ remains unchanged after ρ_k. By the definition of the round, there exists l such that $v = q_l$ ($k < l \leq s$). Since $\mathtt{PC}_{u,v} = PCol_u(\mathtt{Num}_{u,v}) \in \{\mathtt{Num}_{u,v}, \mathtt{Num}_{u,v} + 1, \mathtt{Num}_{u,v} + 2\}$ holds in ρ_{l-1}, v assigns colors to $\mathtt{Col}_v(x)$ ($1 \leq x \leq |Ch_v|$) so that $\mathtt{Col}_v(x) = PCol_v(x)$ should hold for each x ($x \neq \mathtt{Num}_{u,v}$) and $\mathtt{Col}_v(\mathtt{Num}_{u,v}) \in CCol_v(\mathtt{Num}_{u,v})$ should hold in ρ_l. Then, $\mathtt{Col}_v(x)$ ($1 \leq x \leq |Ch_v|$) and $\mathtt{PC}_{u,v}$ are mutually different, and thus, v satisfies $spec(v)$ in ρ_l. After ρ_l, since $\mathtt{PC}_{u,v}$ remains unchanged, v never changes any state. Therefore, the lemma holds.
- Case that $\mathtt{Num}_{u,v} = \mathtt{Num}_{prt_u,u}$ holds. Since $PS(\rho_j, u)$ holds, $\mathtt{PC}_{u,v} = \mathtt{Col}_u(\mathtt{Num}_{u,v}) \in CCol_u(\mathtt{Num}_{u,v}) = \{\mathtt{Num}_{u,v}, \mathtt{Num}_{u,v} + 1, \mathtt{Num}_{u,v} + 2\}$ holds in ρ_j. By the definition of the round, there exists l such that $v = q_l$ ($k < l \leq s$). Since $\mathtt{Col}_v(x) = PCol_v(x)$ for each x ($x \neq \mathtt{Num}_{u,v}$) and $\mathtt{Col}_v(\mathtt{Num}_{u,v}) \in CCol_v(\mathtt{Num}_{u,v})$ hold in ρ_l, $\mathtt{Col}_v(x)$ ($1 \leq x \leq |Ch_v|$) and $\mathtt{PC}_{u,v}$ are different each other, and thus, $SS(\rho_l, v)$ holds.

Next, by the induction, we show that $SS(\rho_m, v)$, $\rho_m|v = \rho_{m+1}|v$, and $\forall r \in Out_v : \rho_m|r = \rho_{m+1}|r$ hold for any configuration ρ_m ($m \geq l$). Assume that $SS(\rho_m, v)$ holds in ρ_m.

- Case that both $q_{m+1} \neq v$ and $q_{m+1} \neq u$ hold. Since v does not execute a step, $\rho_m|v = \rho_{m+1}|v$ and $\forall r \in Out_v : \rho_m|r = \rho_{m+1}|r$ hold. Since $\mathtt{PC}_{u,v}$ remains unchanged, $SS(\rho_{m+1}, v)$ holds.
- Case that $q_{m+1} = v$ holds. Since $\neg En(\rho_m, v)$ holds, $\rho_m|v = \rho_{m+1}|v$ and $\forall r \in Out_v : \rho_m|r = \rho_{m+1}|r$ hold. Since $\mathtt{PC}_{u,v}$ remains unchanged, $SS(\rho_{m+1}, v)$ holds.
- Case that $q_{m+1} = u$ holds. Since v does not execute a step, $\rho_m|v = \rho_{m+1}|v$ and $\forall r \in Out_v : \rho_m|r = \rho_{m+1}|r$ hold. If $\neg En(\rho_m, u)$ holds, $\mathtt{PC}_{u,v}$ remains unchanged, and thus, $SS(\rho_{m+1}, v)$ holds. In the following, we consider the case where $En(\rho_m, u)$ holds. Let $C = \{\mathtt{Num}_{u,v}, \mathtt{Num}_{u,v} + 1, \mathtt{Num}_{u,v} + 2\}$. Since $PS(\rho_m, u)$ holds from Lemma 2, $\mathtt{Col}_u(x) \notin C$ holds for each x ($x \neq \mathtt{Num}_{u,v}$). Furthermore, since $PS(\rho_m, v)$ holds, $\mathtt{Col}_v(x) \notin C$ holds for each x ($x \neq \mathtt{Num}_{u,v}$). Thus, u can assign a color $c \in C - \{\mathtt{PC}_{prt_u,u}, \mathtt{Col}_v(\mathtt{Num}_{u,v})\}$ to $\mathtt{PC}_{u,v}$ ($= \mathtt{Col}_u(\mathtt{Num}_{u,v})$). Thus, $SS(\rho_{m+1}, v)$ holds.

By the induction, for any configuration ρ_m ($m \geq l$), $SS(\rho_m, v)$, $\rho_m|v = \rho_{m+1}|v$, and $\forall r \in Out_v : \rho_m|r = \rho_{m+1}|r$ hold. Therefore, the lemma holds. □

Next, we consider the case B. In the following, for each process p, $ACT(p)$ denotes the last round in which p changes some states of itself or its output registers. If p does not change any state from the initial configuration, we denote $ACT(p) = 0$.

Lemma 4. Let $\mathcal{Q} = q_1, q_2, \ldots$ be a schedule, $e = \rho_0, \rho_1, \ldots$ be an execution by $\mathcal{Q}$. Let v be a nonfaulty process with depth of at least two such that prt_v is a nonfaulty process and prt_{prt_v} is a crashed process. Let ρ_s be the last configuration of the third round. Then, for any configuration ρ_t ($t \geq s$),

i) v satisfies $spec(v)$ in ρ_t, and,
ii) both $\rho_t|v = \rho_{t+1}|v$ and $\forall r \in Out_v : \rho_t|r = \rho_{t+1}|r$ hold.

Proof. Let $u = prt_v$, and x_v be an integer satisfying $v = ch_u(x_v)$. We consider three cases.

Case that $ACT(prt_u) \leq 1$ holds. Since prt_u never changes any state in the second round or later, u never changes any state after u executes a step in the second round. Thus, v never changes any state after v executes a step in the third round. Since u is nonfaulty, $\mathtt{PC}_{u,v} = \mathtt{Col}_u(x_v)$ holds after u executes a step in the first round. Thus, v satisfies $spec(v)$ in the configuration immediately after v executes a step in the third round. Since u and v do not change any state after ρ_s, the lemma holds.

Case that $ACT(prt_u) = 2$ holds. Since prt_u, u, and v behave as nonfaulty processes during the first two rounds, letting ρ_k be the last configuration in the second round, $CS(\rho_k, prt_u)$, $PS(\rho_k, u)$, and $PS(\rho_k, v)$ hold by Lemma 1 and 2. We can show the lemma in the similar way as in the proof of Lemma 3.

Case that $ACT(prt_u) \geq 3$ holds. In this case, prt_u and u behaves as nonfaulty processes during the first three rounds. By Lemma 3, v satisfies $spec(v)$ in the last configuration of third round, and never changes any state after that. □

From Lemmas 3 and 4, proof of Theorem 1 completes.

Corollary 1 states that the protocol LINKCOLORING attains the containment radius of one for crash faults. The containment radius for crash faults is the best possible one in the link register model, when a parent process determines colors of links connecting to its children. This is because, in the initial configuration, a link register can hold an arbitrary value independent of the state of its writer process, and thus, no process can be aware of the colors a parent process assigns to the links connecting to its children when the parent is a crashed process in the initial configuration. For the case that a process crashes after execution of a step, we can show an additional property (Theorem 2) of the protocol LINKCOLORING.

We define that a process v is in a *consistent state* if $\mathtt{PC}_{v,ch_v(x)} = \mathtt{Col}_v(x)$ holds for any x ($1 \leq x \leq |Ch_v|$). Notice that, for any process $v \notin BF$, once v executes an action, v is in a consistent state forever. For nonfaulty process v apart from any Byzantine ancestor by distance of three or more, when prt_v is a crashed process but crashes in a consistent state, we can guarantee that process v satisfies $spec(v)$.

Theorem 2. The protocol LINKCOLORING is a $(\Delta + 1)$-link-coloring protocol satisfying the following property:

- Let $e = \rho_0, \rho_1, \ldots$ be any execution, $v \in P$ be any nonfaulty process v, and ρ_s be the last configuration of the third round. When v has no Byzantine ancestor with the distance of two or less and has the crashed parent that crashes in a consistent state, for any t ($t \geq s$),
  i) v satisfies $spec(v)$ in ρ_t, and,
  ii) both $\rho_t|v = \rho_{t+1}|v$ and $\forall r \in Out_v : \rho_t|r = \rho_{t+1}|r$ holds.

In what follows, we prove Theorem 2. We show Theorem 2 by considering the following two cases:

A. Process v is nonfaulty, and prt_v is the root process and crashes in a consistent state (Lemma 5).
B. Process v is nonfaulty, prt_v crashes in a consistent state, and prt_{prt_v} is not a Byzantine process (Lemma 6).

First, we consider the case A.

Lemma 5. Let $\mathcal{Q} = q_1, q_2, \ldots$ be a schedule and $e = \rho_0, \rho_1, \ldots$ be an execution by $\mathcal{Q}$. Let v be a nonfaulty process such that prt_v is the root process. Assume that prt_v crashes in a consistent state. Let ρ_s be the last configuration of the second round. Then, for any configuration ρ_t ($t \geq s$),

i) v satisfies $spec(v)$ in ρ_t, and,
ii) both $\rho_t|v = \rho_{t+1}|v$ and $\forall r \in Out_v : \rho_t|r = \rho_{t+1}|r$ hold.

Proof. Let $u = prt_v$ and x_v be an integer satisfying $v = ch_{prt_v}(x_v)$. We consider two cases.

Case that u executes a step. By the protocol, u never changes any state after u executes its first step. Consequently, $\mathtt{PC}_{u,v}$ remains unchanged after the last configuration of the first round. Notice that $\mathtt{PC}_{u,v} = \mathtt{Col}_u(x_v)$ holds. Thus, once v executes a step in the second round, v satisfies $spec(v)$ and never changes any state after that.

Case that u does not execute a step. Then, $\mathtt{PC}_{u,v}$ remains unchanged forever. Since u crashes in a consistent state, $\mathtt{PC}_{u,v} = \mathtt{Col}_u(x_v)$ holds. Thus, once v executes a step in the second round, v satisfies $spec(v)$ and never changes any state after that. □

Next, we consider the case B.

Lemma 6. Let $\mathcal{Q} = q_1, q_2, \ldots$ be a schedule and $e = \rho_0, \rho_1, \ldots$ be an execution by $\mathcal{Q}$. Let v be a nonfaulty process with depth of at least two. Assume that prt_v crashes in a consistent state and $prt_{prt_v} \notin BF$. Let ρ_s be the last configuration of the third round. Then, for any configuration ρ_t ($t \geq s$),

i) v satisfies $spec(v)$ in ρ_t, and,
ii) both $\rho_t|v = \rho_{t+1}|v$ and $\forall r \in Out_v : \rho_t|r = \rho_{t+1}|r$ hold.

Proof. Let $u = prt_v$ and x_v be an integer satisfying $v = ch_u(x_v)$. We consider two cases.

Case that $ACT(u) \leq 2$ holds. Since u never changes any state in the third round or later, v never changes any state after v executes a step in the third round. Since u crashes in a consistent state, $\mathtt{PC}_{u,v} = \mathtt{Col}_u(x_v)$ holds after the last configuration of the second round. Thus, v satisfies $spec(v)$ in the configuration immediately after v executes a step in the third round. Since u and v do not change any state after ρ_s, the lemma holds.

Case that $ACT(u) \geq 3$ holds. In this case, u behaves as nonfaulty processes during the first three rounds. By Theorem 1, v satisfies $spec(v)$ in the last configuration of third round, and never changes any state after that. □

From Theorem 1, Lemmas 5 and 6, Theorem 2 holds.

IV. Impossibility of Link-Coloring Using Δ Colors under the Central Daemon

In this section, we consider a self-stabilizing Δ-link-coloring protocol. For any tree network, Δ-link-coloring is possible. However, we show that, for any self-stabilizing Δ-link-coloring protocol, the containment radius is $\Omega(\log n)$ if $\Delta \geq 3$, and $\Omega(n)$ if $\Delta = 2$, where n is the number of processes. Thus, the protocol LINKCOLORING attains the minimality in the number of colors for achieving a constant containment radius for Byzantine faults. To show the lower bounds, we define a *view* of v as the states of v and all link registers in In_v. In the following discussion, $view(\rho, v)$ denotes the view of process v in configuration ρ.

Theorem 3. Assume $\Delta \geq 3$. For any $(\mathcal{B}, \mathcal{C})$-self-stabilizing Δ-link-coloring protocol with containment radius (τ, μ) in tree networks, $\tau = \Omega(\log n)$ holds, where n is the number of processes.

In what follows, we prove Theorem 3 by contradiction. We assume that the color of a link (u, v) is determined (or coded) by the states of u, v, $r_{u,v}$, and $r_{v,u}$. Assume that A is a $(\mathcal{B}, \mathcal{C})$-self-stabilizing Δ-link-coloring protocol with containment radius (τ, μ).

Let a system $S = (P, L)$ be a complete $(\Delta - 1)$-ary tree such that each non-leaf process has exactly $\Delta - 1$ children and all leaf processes are at the same depth, say h. Then, $n = \sum_{k=0}^{h} (\Delta - 1)^k$ holds. Let $P = \{v_1, v_2, \ldots, v_n\}$, and let $ch^S_{v_i}(x)$ be the x-th child of v_i in S. Let v_l be a process with depth of $\lceil h/2 \rceil$, $v_m = prt_{v_l}$, and c be an integer satisfying $v_l = ch^S_{v_m}(c)$. We assume the set of Byzantine processes $BF = \{v_l\}$ and the set of crashed processes $CF = \emptyset$.

First, we claim the following.

- From any configuration, behavior of Byzantine process v_l can eventually lead the system S to the configuration satisfying the following Condition $\mathcal{A}$ (We show this claim in Lemma 7).

Condition $\mathcal{A}$:

1) For any $v_i \in P - \{v_m, v_l\}$, all links incident to v_i have different colors.
2) Letting E_l and E_m be the sets of all links incident to v_l and v_m respectively, $\{Color(e) \mid e \in (E_l \cup E_m) - \{(v_l, v_m)\}\} = CSET$ holds.

Notice that, in the configuration satisfying Condition $\mathcal{A}$, whatever color is assigned to link (v_l, v_m), a link incident to v_l or v_m has the same color as link (v_l, v_m).

- From any configuration satisfying Condition $\mathcal{A}$, behavior of Byzantine process v_l can make some process apart from v_l at distance $\Omega(\log n)$ eventually change its state (We show this claim in Lemma 8).

When the above executions are alternatively repeated in infinitely many times, some processes apart from v_l at distance $\Omega(\log n)$ change their states infinitely often. Therefore, $\tau = \Omega(\log n)$ holds. In the followings, we show the above claims.

Lemma 7. From any configuration, behavior of Byzantine process v_l can eventually lead the system S to the configuration satisfying Condition $\mathcal{A}$.

Proof. Let ρ_0 be any configuration of system S. To make the execution such that system S eventually reaches the configuration satisfying Condition $\mathcal{A}$, we construct a system $T = (P', L')$ as follows. Let $P' = P \cup \{u_1, u_2, \ldots, u_{\Delta-1}\}$, where $u_i \notin P$ $(1 \leq i \leq \Delta - 1)$, and L' is defined as follows (See Fig. 6):

1. $ch^T_{v_i}(x) = ch^S_{v_i}(x)$ $(i \neq m)$
2. $ch^T_{v_m}(x) = \begin{cases} u_1 & (x = c) \\ ch^S_{v_m}(x) & (x \neq c) \end{cases}$
3. $ch^T_{u_1}(x) = \begin{cases} u_{x+1} & (x < c) \\ v_l & (x = c) \\ u_x & (x > c) \end{cases}$

For the system T, let σ_0 be the initial configuration such that $view(\rho_0, v_i) = view(\sigma_0, v_i)$ holds for any v_i, $\mathcal{Q}_T = p_1, p_2, \ldots$ be a schedule, $e_T = \sigma_0, \sigma_1, \ldots$ be the execution by schedule $\mathcal{Q}_T$ in the case that $BF = \emptyset$ and $CF = \emptyset$. Then, in e_T, there exists a configuration σ_s such that any process v satisfies $spec(v)$ and never changes any state after σ_s.

Next, for the system S, we construct a schedule $\mathcal{Q}_S = q_1, q_2, \ldots$ and an execution $e_S = \rho_0, \rho_1, \ldots$ in the case that $BF = \{v_l\}$ and $CF = \emptyset$. We construct the execution e_S such that Byzantine process v_l simulates the behavior of u_1 and v_l in e_T (See Fig. 7). Formally, we construct $\mathcal{Q}_S$ and e_S as follows:

We construct the schedule $\mathcal{Q}_S$ in which v_i $(i \neq l)$ executes a step in the same order as in $\mathcal{Q}_T$, and v_l executes a step immediately before each step of neighbors of v_l. We construct the execution e_S such that Byzantine process v_l simulates the behavior of u_1 in e_T for its parent v_m and simulates v_l in e_T for its children. That is, if $q_\alpha = v_l, q_{\alpha+1} = v_m$, and $q_{\alpha+1}$ is the k-th step of v_m in e_S, v_l changes the states of itself and link register r_{v_l,v_m} so that $\rho_\alpha|v_l = \sigma_\beta|v_l$ and $view(\rho_\alpha, v_m) = view(\sigma_\beta, v_m)$, where σ_β is the configuration such that $p_{\beta+1}$ is the k-th step of v_m in e_T. And, if $q_\alpha = v_l$, $q_{\alpha+1} = ch^S_{v_l}(x)$, and $q_{\alpha+1}$ is the k-th step of $ch^S_{v_l}(x)$ in e_S, v_l changes the states of itself and its output

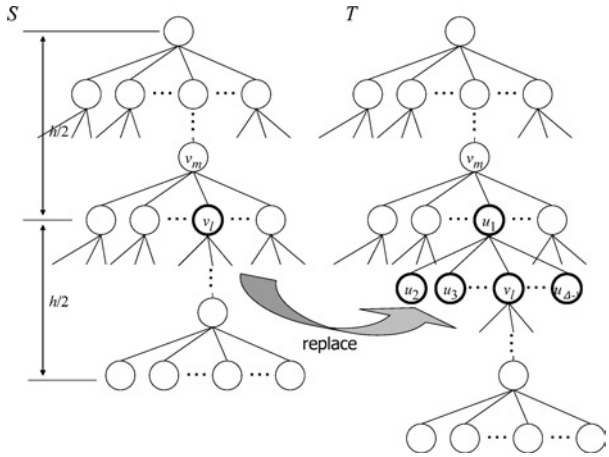


Fig. 6 Two systems.

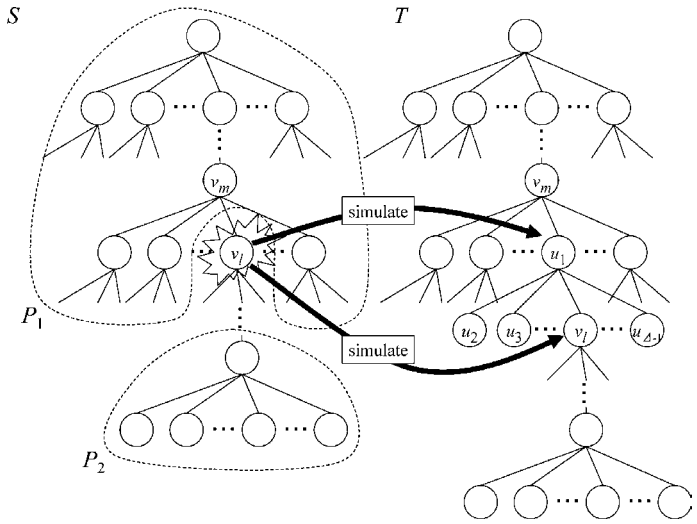


Fig. 7 Byzantine process v_l in S behaves as u_1 for P_1 and as v_l for P_2.

registers so that $\rho_\alpha|v_l = \sigma_\beta|v_l$ and $view(\rho_\alpha, ch^S_{v_l}(x)) = view(\sigma_\beta, ch^T_{v_l}(x))$, where σ_β is the configuration such that $p_{\beta+1}$ is the k-th step of $ch^T_{v_l}(x)$ in e_T.

Then, any v_i $(i \neq l)$ changes the states of itself and its output registers in the same way in e_T. By the definition of e_S and e_T, there exists a configuration ρ_t such that, any process v_i $(i \neq l)$ never changes its state after ρ_t.

Let t' be the integer such that $t' \geq t$ and $q_{t'+1} \in Ch_{v_l}$. In $\rho_{t'}$, colors of links not connecting to v_l are the same as those in σ_s. In addition, since $v_l(= q_{t'})$ simulates v_l in T, the state of v_l in $\rho_{t'}$ is the same as that in σ_s. Thus, colors of links between v_l and its children are the same as those in σ_s. Therefore, colors of links except for (v_m, v_l) in $\rho_{t'}$ are the same as those in σ_s, and thus, $\rho_{t'}$ satisfies the first condition of $\mathcal{A}$.

In the following, we show that $\rho_{t'}$ also satisfies the second condition of $\mathcal{A}$. Let $E_S(v)$ and $E_T(v)$ be the sets of all links incident to v in S and in T respectively. Let $COL(\rho, E)$ be the set of colors that links in E have in configuration ρ. Let $U = COL(\rho_{t'}, (E_S(v_l) \cup E_S(v_m)) - \{(v_l, v_m)\})$, $V = COL(\sigma_s, E_T(v_l) - \{(v_l, u_1)\})$, and $W = COL(\sigma_s, E_T(v_m) - \{(v_m, u_1)\})$. By the definition of $\rho_{t'}$, $U = V \cup W$ holds. Then, since degrees of v_l and v_m in T are Δ, letting χ_1 and χ_2 be the colors of (v_l, u_1) and (v_m, u_1) in configuration σ_s, $V = CSET - \{\chi_1\}$ and $W = CSET - \{\chi_2\}$ hold. Since $\chi_1 \neq \chi_2$, $V \cup W = CSET$ holds, and thus, $U = CSET$ holds. Therefore, $\rho_{t'}$ satisfies the second condition of $\mathcal{A}$. □

Lemma 8. From any configuration satisfying Condition $\mathcal{A}$, behavior of Byzantine process v_l can make some process apart from v_l at distance $\Omega(\log n)$ eventually change its state.

Proof. Let ρ be a configuration satisfying $\mathcal{A}$. We consider the execution from ρ such that a Byzantine process v_l behaves as a nonfaulty process, that is, all processes behave correctly. The execution is the same as the execution from ρ in the case all processes are nonfaulty. Thus, the system reaches a configuration ρ' where any process v satisfies $spec(v)$. Recall that, in ρ, whatever color is assigned to link (v_l, v_m), a link incident to v_l or v_m has the same color as link (v_l, v_m). Thus, there exists $v_{a_1} \in \{v_l, v_m\}$ and $v_{a_2} \in N_{v_{a_1}} - \{v_l, v_m\}$ such that the color of link (v_{a_1}, v_{a_2}) in ρ' is differnt from that in ρ. Since all links incident to v_{a_2} have different colors in ρ, if the degree of v_{a_2} is Δ, there exists neighbor v_{a_3} of v_{a_2} such that the color of link (v_{a_2}, v_{a_3}) in ρ' is different from that in ρ. In the similar way, we can construct a sequence of links, $(v_{a_1}, v_{a_2}), (v_{a_2}, v_{a_3}), \ldots, (v_{a_{f-1}}, v_{a_f})$, whose colors in ρ' are different from those in ρ. Notice that the degree of v_{a_f}, the termination process in the sequence, is not Δ. Consequently, the state of either $v_{a_{f-1}}$, v_{a_f}, r_{a_{f-1},a_f}, or $r_{a_f,a_{f-1}}$ in ρ is different from that in ρ'. Since the degree of v_{a_f} is not Δ and the system S is a complete $(\Delta - 1)$-ary tree, v_{a_f} is the root process or a leaf process. Thus, the distance from $v_{a_{f-1}}$ (or v_{a_f}) to a

Byzantine process v_l is $\Omega(h) = \Omega(\log n)$. Since ρ is a $(\mathcal{B}, \mathcal{C})$-stable configuration with containment radius (τ, μ), processes whose distance to v_l is more than τ do not change any state. Therefore, $\tau = \Omega(\log n)$. □

From Lemmas 7 and 8, Theorem 3 holds.

Similarly, we can prove the following theorem for the case of $\Delta = 2$, i.e., a line graph of processes.

Theorem 4. Assume $\Delta = 2$. For any $(\mathcal{B}, \mathcal{C})$-self-stabilizing Δ-link-coloring protocol with containment radius (τ, μ) in tree networks, $\tau = \Omega(n)$ holds, where n is the number of processes.

V. Impossibility of Link-Coloring under the Distributed Daemon

In this section, we consider a self-stabilizing link-coloring protocol under the distributed daemon. The distributed daemon allows two or more processes to execute their actions simultaneously, while the central daemon does not. Thus, the distributed daemon is usually regarded as a more practical model. However in this section, we show that, for any self-stabilizing link-coloring protocol, the influence of a Byzantine process expands to a process whose distance to the Byzantine process is $\Omega(n)$ even when it can use arbitrarily large number of colors. This lower bound result implies that the assumption of the central daemon is reasonable to attain a constant containment radius for Byzantine faults.

Theorem 5. Let n be the number of processes, and A be a link-coloring protocol under the distributed daemon. If A is a $(\mathcal{B}, \mathcal{C})$-self-stabilizing protocol with containment radius (τ, μ), $\tau = \Omega(n)$ holds.

Proof. We assume that the color of a link (u, v) is determined by the states of u, v, $r_{u,v}$, and $r_{v,u}$. Assume that A is a $(\mathcal{B}, \mathcal{C})$-self-stabilizing protocol with containment radius (τ, μ).

We consider a distributed system $S = (P, L)$ in the form of a line graph: Let $P = \{v_1, v_2, \ldots, v_n\}$ and $L = \{(v_i, v_{i+1}) | 1 \leq i \leq n-1\}$, where v_1 is the root process. Let $BF = \{v_1, v_n\}$ and $CF = \emptyset$.

We consider an execution $e = \rho_0, \rho_1, \ldots$ by a schedule $\mathcal{Q} = Q_1, Q_2, \ldots$, where $Q_i = P$ for any i. Consider the initial configuration ρ_0 such that $view(\rho_0, v_2) = view(\rho_0, v_3) = \cdots = view(\rho_0, v_{n-1})$ holds. Since processes $v_2, v_3, \ldots, v_{n-1}$ execute the same step, $view(\rho_1, v_3) = view(\rho_1, v_4) = \cdots = view(\rho_1, v_{n-2})$ holds (say $view(\rho_1, v_3) = s_1$). Then, we consider that Byzantine processes v_1 and v_n change the states so that $view(\rho_1, v_2) = view(\rho_1, v_{n-1}) = s_1$ can hold. This implies $view(\rho_1, v_2) = view(\rho_1, v_3) = \cdots = view(\rho_1, v_{n-1}) = s_1$ holds. When Byzantine processes execute their steps similarly, we have $view(\rho_i, v_2) = view(\rho_i, v_3) = \cdots = view(\rho_i, v_{n-1}) = s_i$ for any i. It shows that, for any ρ_i, $Color((v_2, v_3)) = Color((v_3, v_4)) = \cdots = Color((v_{n-2}, v_{n-1}))$ holds. Thus, letting $h = \lceil \frac{n}{2} \rceil$, a process v_h cannot satisfy $spec(v_h)$. Since the distance from v_h to any Byzantine process is $\Omega(n)$, $\tau = \Omega(n)$ holds. □

VI. Conclusion

In this paper, we considered a self-stabilizing link-coloring protocol resilient to Byzantine faults in rooted tree networks. First, under the central daemon, we proposed a self-stabilizing link-coloring protocol resilient to Byzantine faults. The protocol uses $\Delta + 1$ colors, where Δ is the maximum degree of the network, and guarantees that any nonfaulty process v reaches its desired states within three rounds and never changes its state after that if v has no Byzantine ancestor with the distance of two or less and has no crashed parent.

We also show the following lower bound results for any self-stabilizing link-coloring protocol resilient to Byzantine faults:

1. When only Δ colors are used, the containment radius becomes $\Omega(\log n)$ if $\Delta \geq 3$, and $\Omega(n)$ if $\Delta = 2$, where n is the number of processes.
2. For any self-stabilizing link-coloring protocol that assumes the distributed daemon, the containment radius is $\Omega(n)$ even when it can use arbitrary number of colors.

These two lower bound results prove necessity of $\Delta + 1$ colors and the central daemon to attain the fault containment against Byzantine faults with a constant containment radius.

Appendix A The Table of Notations

Table 1 The table of notations introduced in Section 2.

Notation	Meaning
N_v	The set of neighbors of v.
prt_v	The parent of v.
Ch_v	The set of children of v.
$ch_v(x)$	The x-th child of v.
Δ	The maximum degree of a tree network.
In_v	The input register set of v.
Out_v	The output register set of v.
$\rho\|v$ ($\rho\|r$)	The state of process v (link register r) at configuration ρ.
$En(\rho, v)$	The predicate such that $En(\rho, v) = true$ iff v is enabled at configuration ρ.
BF	The set of Byzantine processes.
CF	The set of crashed processes.

Table 2 The table of notations used in Section 3, where $u = prt_v$ and $c_x = ch_v(x)$ ($1 \leq x \leq |Ch_v|$).

Notation	Meaning
$\mathtt{Col}_v(x)$	The variable on v that represents a color of link (v, c_x).
$\mathtt{Num}_{u,v}$	The variable on link register $r_{u,v}$. When v is the x-th child of u, $\mathtt{Num}_{u,v}$ is set to x.
$\mathtt{PC}_{u,v}$	The variable on link register $r_{u,v}$ that represents a color of link (u, v).
$\mathtt{USET}_{v,u}$	The variable on link register $r_{v,u}$ that represents $\{\mathtt{Col}_v(x) \| 1 \leq x \leq \|Ch_v\|\}$.
$CCol_v(x)$	The candidate color set of link (v, c_x).
$PCol_v(x)$	The preference color of link (v, c_x).
$CS(\rho, v)$	The predicate such that $CS(\rho, v) = true$ iff v is in a candidate state at configuration ρ.
$PS(\rho, v)$	The predicate such that $PS(\rho, v) = true$ iff v is in a preference state at configuration ρ.
$SS(\rho, v)$	The predicate such that $SS(\rho, v) = true$ iff v is in a stable state at configuration ρ.
$ACT(v)$	The last round in which v changes some states of itself or its output registers.

Acknowledgements

This work is supported in part by MEXT: "The 21st Century Center of Excellence Program", JSPS: Grant-in-Aid for Scientific Research ((B)15300017), and MEXT: Grant-in-Aid for Scientific Research on Priority Areas (16092215).

References

[1]Dijkstra, E. W., "Self Stabilizing Systems in Spite of Distributed Control," *Communications of the Association of the Computing Machinery*, Vol. 17, 1974, pp. 643–644.

[2]Dolev, S., "*Self-Stabilization*," MIT Press, 2000.

[3]Anagnostou, E., and Hadzilacos, V., "Tolerating Transient and Permanent Failures," *Lectures Notes in Computer Science*, Vol. 725, Springer-Verlag, 1993, pp. 174–188.

[4]Beauquier, J., and Kekkonen-Moneta, S., "Fault-Tolerance and Self-Stabilization: Impossibility Results and Solutions using Self-Stabiling Failure Detectors," *International Journal of Systems Science*, Vol. 28, No. 11, 1997, pp. 1177–1187.

[5]Beauquier, J., and Kekkonen-Moneta, S., "On Ftss-Solvable Distributed Problems," *Proceedings of the 6^{th} Annual ACM Symposium on Principles of Distributed Computing*, 1997, p. 290.

[6]Gopal, A. S., and Perry, K. J., "Unifying Self-Stabilization and Fault-Tolerance," *Proceedings of the 12^{th} Annual ACM Symposium on Principles of Distributed Computing*, 1993, pp. 195–206.

[7]Masuzawa, T., "A Fault-Tolerant and Self-Stabilizing Protocol for the Topology Problem," *Proceedings of the 2^{nd} Workshop on Self-Stabilizing Systems*, 1995, pp. 1.1–1.15.

[8]Matsui, H., Inoue, M., Masuzawa, T., and Fujiwara, H., "Fault-Tolerant and Self-Stabilizing Protocols using an Unreliable Failure Detector," *IEICE Transactions on Information and Systems*, Vol. E83-D, No. 10, 2000, pp. 1831–1840.

[9]Nesterenko, M., and Arora, A., "Tolerance to Unbounded Byzantine Faults," *Proceedings of 21st IEEE Symposium on Reliable Distributed Systems*, 2002, pp. 22–29.

[10]Ukena, S., Katayama, Y., Masuzawa, T., and Fujiwara, H., "A Self-Stabilizing Spanning Tree Protocol that Tolerates Non-Quiescent Permanent Faults. *IEICE Transaction*, Vol. J85-D-I, No. 11, 2002, pp. 1007–1014.

[11]Ghosh, S., and Gupta, A. "An Exercise in Fault-Containment: Self-Stabilizing Leader Election," *Information Processing Letters*, Vol. 59, No. 5, 1996, pp. 281–288.

[12]Ghosh, S., Gupta, A., Herman, T., and Pemmaraju, S. V., "Fault-Containing Self-Stabilizing Algorithms," *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, 1996, pp. 45–54.

[13]Ghosh, S., and Pemmaraju, S. V., "Tradeoffs in Fault-Containing Self-Stabilization," *Proceedings of the 3rd Workshop on Self-Stabilizing Systems*, 1997, pp. 157–169.

[14]Kutten, S., and Patt-Shamir, B., "Stabilizing Time-Adaptive Protocols," *Theoretical Computer Science*, Vol. 220, No. 1, 1999, pp. 93–111.

[15]Katayama, Y., and Masuzawa, T., "A Fault-Containing Self-Stabilizing Protocol for Constructing a Minimum Spanning Tree," *IEICE Transactions*, Vol. J84-D-I, No. 9, 2001, pp. 1307–1317.

[16]Arora, A., and Zhang, H., "Lsrp: Local Stabilization in Shortest Path Routing," *Proceedings of the 2003 International Conference on Dependable Systems and Networks*, 2003, pp. 139–148.

[17]Beaumont, O., and Marchal, L., "Pipelining Broadcasts on Heterogeneous Platforms Under the One-Port Model," *Research Report RR-2004-32, LIP, ENS Lyon, France*, 2004.

[18]Grable, D. A., and Panconesi, A., "Nearly Optimal Distributed Edge Colouring in o(log log n) Rounds," *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1997, pp. 278–285.

[19]Marathe, M. V. Panconesi, A., and Risinger, L. D., "An Experimental Study of a Simple, Distributed Edge Coloring Algorithm," *Proceedings of the 12th Annual ACM Symposium on Parallel Algorithms and Architectures*, 2000, pp. 166–175.

[20]Panconesi, A., and Srinivasan, A., "Fast Randomized Algorithms for Distributed Edge Coloring," *Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing*, 1992, pp. 251–262.

Shlomi Dolev
Associate Editor